Red Hat

OpenShift Container Platform 4.10

Telco RAN 4.10 Developer Preview features

2022-05-04

Developer Preview statement . . . 1
OpenShift Container Platform 4.10 release notes . . . 2
About this release . . . 2
OpenShift Container Platform layered and dependent component support and compatibility . . . 2
New features and enhancements . . . 2
Notable technical changes . . . 34
Deprecated and removed features . . . 35
Bug fixes . . . 41
Technology Preview features . . . 62
Known issues . . . 64
Asynchronous errata updates . . . 70
Monitoring Redfish hardware events . . . 77
About Redfish hardware events . . . 77
How Redfish hardware events work . . . 77
Installing the upstream Hardware Event Proxy Operator . . . 79
Installing the AMQ messaging bus . . . 79
Subscribing to Redfish BMC hardware events for a cluster node . . . 80
Subscribing applications to Redfish hardware event notifications REST API reference . . . 87
Cluster Group Upgrades Operator for cluster updates . . . 91
About the Cluster Group Upgrades Operator configuration . . . 91
About managed policies used with Cluster Group Upgrades Operator . . . 91
Installing the upstream Cluster Group Upgrades Operator . . . 92
About the ClusterGroupUpgrade CR . . . 92
Update policies on managed clusters . . . 107
Using the container image pre-cache feature . . . 115
Troubleshooting the Cluster Group Upgrades Operator . . . 119
Deploying distributed units manually on single node OpenShift . . . 129
Configuring the distributed units (DUs) . . . 129
Applying the distributed unit (DU) configuration to a single node cluster . . . 146
Deploying distributed units at scale in a disconnected environment . . . 147
Provisioning edge sites at scale . . . 147
About ZTP and distributed units on OpenShift clusters . . . 148
The GitOps approach . . . 149
Zero touch provisioning building blocks . . . 149
How to plan your RAN policies . . . 150
Low latency for distributed units (DUs) . . . 151
Preparing the disconnected environment . . . 151
Installing Red Hat Advanced Cluster Management in a disconnected environment . . . 153
Enabling assisted installer service on bare metal . . . 154
ZTP custom resources . . . 156

PolicyGenTemplate CRs for RAN deployments . . . 158
About the PolicyGenTemplate . . . 159
Best practices when customizing PolicyGenTemplate CRs . . . 162
Creating the PolicyGenTemplate CR . . . 163
Creating ZTP custom resources for multiple managed clusters . . . 164
Installing the GitOps ZTP pipeline . . . 172
Adding new content to the GitOps ZTP pipeline . . . 175
Customizing extra installation manifests in the ZTP GitOps pipeline . . . 177
Deploying a site . . . 177
GitOps ZTP and Cluster Group Upgrades Operator (CGUO) . . . 182
Monitoring deployment progress . . . 183
Indication of done for ZTP installations . . . 184
Troubleshooting GitOps ZTP . . . 188
Site cleanup . . . 193
Upgrading GitOps ZTP . . . 194
Manually install a single managed cluster . . . 199
Generating RAN policies . . . 212
Updating managed policies with the Cluster Group Upgrades Operator . . . 216
End-to-end procedures for updating clusters in a disconnected environment . . . 217

Developer Preview statement

Developer Preview features are not supported with Red Hat production service level agreements (SLAs) and are not functionally complete or production-ready. Do not use Developer Preview features for production or business-critical workloads. Developer Preview features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. These features might not have any documentation, and testing is limited. Red Hat might provide ways to submit feedback on Developer Preview features without an associated SLA.

OpenShift Container Platform 4.10 release notes

Red Hat OpenShift Container Platform provides developers and IT organizations with a hybrid cloud application platform for deploying both new and existing applications on secure, scalable resources with minimal configuration and management overhead. OpenShift Container Platform supports a wide selection of programming languages and frameworks, such as Java, JavaScript, Python, Ruby, and PHP.

Built on Red Hat Enterprise Linux (RHEL) and Kubernetes, OpenShift Container Platform provides a more secure and scalable multitenant operating system for today's enterprise-class applications, while delivering integrated application runtimes and libraries. OpenShift Container Platform enables organizations to meet security, privacy, compliance, and governance requirements.

About this release

OpenShift Container Platform (RHSA-2022:0056) is now available. This release uses Kubernetes 1.23 with CRI-O runtime. New features, changes, and known issues that pertain to OpenShift Container Platform 4.10 are included in this topic.

Red Hat did not publicly release OpenShift Container Platform 4.10.0 as the GA version and, instead, is releasing OpenShift Container Platform 4.10.3 as the GA version.

OpenShift Container Platform 4.10 clusters are available at https://console.redhat.com/openshift. The Red Hat OpenShift Cluster Manager application for OpenShift Container Platform allows you to deploy OpenShift clusters to either on-premise or cloud environments.

OpenShift Container Platform 4.10 is supported on Red Hat Enterprise Linux (RHEL) 8.4 and 8.5, as well as on Red Hat Enterprise Linux CoreOS (RHCOS) 4.10.

You must use RHCOS machines for the control plane, and you can use either RHCOS or RHEL for compute machines.

OpenShift Container Platform layered and dependent component support and compatibility

The scope of support for layered and dependent components of OpenShift Container Platform changes independently of the OpenShift Container Platform version. To determine the current support status and compatibility for an add-on, refer to its release notes. For more information, see the Red Hat OpenShift Container Platform Life Cycle Policy.

New features and enhancements

This release adds improvements related to the following components and concepts.

Documentation

Getting started with OpenShift Container Platform

OpenShift Container Platform 4.10 now includes a getting started guide. Getting Started with OpenShift Container Platform defines basic terminology and provides role-based next steps for developers and administrators.

The tutorials walk new users through the web console and the OpenShift CLI (`oc`) interfaces. New users can accomplish the following tasks through the Getting Started:

- Create a project
- Grant view permissions
- Deploy a container image from Quay
- Examine and scale an application
- Deploy a Python application from GitHub
- Connect to a database from Quay
- Create a secret
- Load and view your application

For more information, see Getting Started with OpenShift Container Platform.

Red Hat Enterprise Linux CoreOS (RHCOS)

Improved customization of bare metal RHCOS installation

The `coreos-installer` utility now has `iso customize` and `pxe customize` subcommands for more flexible customization when installing RHCOS on bare metal from the live ISO and PXE images.

This includes the ability to customize the installation to fetch Ignition configs from HTTPS servers that use a custom certificate authority or self-signed certificate.

Installation and upgrade

New default component types for AWS installations

The OpenShift Container Platform 4.10 installer uses new default component types for installations on AWS. The installation program uses the following components by default:

- AWS EC2 M6i instances for both control plane and compute nodes, where available
- AWS EBS gp3 storage

Enhancements to API in the `install-config.yaml` file

Previously, when a user installed OpenShift Container Platform on a bare metal installer-provisioned infrastructure, they had nowhere to configure custom network interfaces, such as static IPs or vLANs to communicate with the Ironic server.

When configuring a Day 1 installation on bare metal only, users can now use the API in the `install-config.yaml` file to customize the network configuration (`networkConfig`). This configuration is set during the installation and provisioning process and includes advanced options, such as setting static IPs per host.

OpenShift Container Platform on ARM

OpenShift Container Platform 4.10 is now supported on ARM based AWS EC2 and bare-metal platforms. Instance availability and installation documentation can be found in Supported installation methods for different platforms.

The following features are supported for OpenShift Container Platform on ARM:

- OpenShift Cluster Monitoring
- RHEL 8 Application Streams
- OVNKube
- Elastic Block Store (EBS) for AWS
- AWS .NET applications
- NFS storage on bare metal

The following Operators are supported for OpenShift Container Platform on ARM:

- Node Tuning Operator
- Node Feature Discovery Operator
- Cluster Samples Operator
- Cluster Logging Operator
- Elasticsearch Operator
- Service Binding Operator

Installing a cluster on IBM Cloud using installer-provisioned infrastructure (Technology Preview)

OpenShift Container Platform 4.10 introduces support for installing a cluster on IBM Cloud using installer-provisioned infrastructure in Technology Preview.

The following limitations apply for IBM Cloud using IPI:

- Deploying IBM Cloud using IPI on a previously existing network is not supported.
- The Cloud Credential Operator (CCO) can use only Manual mode. Mint mode or STS are not supported.
- IBM Cloud DNS Services is not supported. An instance of IBM Cloud Internet Services is required.
- Private or disconnected deployments are not supported.

For more information, see Preparing to install on IBM Cloud.

Thin provisioning support for VMware vSphere cluster installation

OpenShift Container Platform 4.10 introduces support for thin-provisioned disks when you install a cluster using installer-provisioned infrastructure. You can provision disks as `thin`, `thick`, or `eagerZeroedThick`. For more information about disk provisioning modes in VMware vSphere, see Installation configuration parameters.

Installing a cluster into an Amazon Web Services GovCloud region

Red Hat Enterprise Linux CoreOS (RHCOS) Amazon Machine Images (AMIs) are now available for AWS GovCloud regions. The availability of these AMIs improves the installation process because you are no longer required to upload a custom RHCOS AMI to deploy a cluster.

For more information, see Installing a cluster on AWS into a government region.

Using a custom AWS IAM role for instance profiles

Beginning with OpenShift Container Platform 4.10, if you configure a cluster with an existing IAM role, the installation program no longer adds the `shared` tag to the role when deploying the cluster. This enhancement improves the installation process for organizations that want to use a custom IAM role, but whose security policies prevent the use of the `shared` tag.

CSI driver installation on vSphere clusters

To install a CSI driver on a cluster running on vSphere, you must have the following components installed:

- Virtual hardware version 15 or later
- vSphere version 6.7 Update 3 or later
- VMware ESXi version 6.7 Update 3 or later

Components with versions earlier than those above are still supported, but are deprecated. These versions are still fully supported, but version 4.11 of OpenShift Container Platform will require vSphere virtual hardware version 15 or later.

If your cluster is deployed on vSphere, and the preceding components are lower than the version mentioned above, upgrading from OpenShift Container Platform 4.9 to 4.10 on vSphere is supported, but no vSphere CSI driver will be installed. Bug fixes and other upgrades to 4.10 are still supported, however upgrading to 4.11 will be unavailable.

Installing a cluster on Alibaba Cloud using installer-provisioned infrastructure (Technology Preview)

OpenShift Container Platform 4.10 introduces the ability for installing a cluster on Alibaba Cloud using installer-provisioned infrastructure in Technology Preview. This type of installation lets you use the installation program to deploy a cluster on infrastructure that the installation program provisions and the cluster maintains.

Installing a cluster on Microsoft Azure Stack Hub using installer-provisioned infrastructure

OpenShift Container Platform 4.10 introduces support for installing a cluster on Azure Stack Hub using installer-provisioned infrastructure. This type of installation lets you use the installation program to deploy a cluster on infrastructure that the installation program provisions and the cluster maintains.

For more information, see Installing a cluster on Azure Stack Hub with an installer-provisioned infrastructure.

Conditional updates

OpenShift Container Platform 4.10 adds support for consuming conditional update paths provided by the OpenShift Update Service. Conditional update paths convey identified risks and the conditions under which those risks apply to clusters. The Administrator perspective on the web console only offers recommended upgrade paths for which the cluster does not match known risks. However, OpenShift CLI (`oc`) 4.10 or later can be used to display additional upgrade paths for OpenShift Container Platform 4.10 clusters. Associated risk information including supporting documentation references is displayed with the paths. The administrator may review the referenced materials and choose to perform the supported, but no longer recommended, upgrade.

For more information, see Conditional updates and Updating along a conditional upgrade path.

Disconnected mirroring with the oc-mirror CLI plug-in (Technology Preview)

This release introduces the oc-mirror OpenShift CLI (`oc`) plug-in as a Technology Preview. You can use the oc-mirror plug-in to mirror images in a disconnected environment.

For more information, see Mirroring images for a disconnected installation using the oc-mirror plug-in.

Installing a cluster on RHOSP that uses OVS-DPDK

You can now install a cluster on Red Hat OpenStack Platform (RHOSP) for which compute machines run on Open vSwitch with the Data Plane Development Kit (OVS-DPDK) networks. Workloads that run on these machines can benefit from the performance and latency improvements of OVS-DPDK.

For more information, see Installing a cluster on RHOSP that supports DPDK-connected compute machines.

Setting compute machine affinity during installation on RHOSP

You can now select compute machine affinity when you install a cluster on RHOSP. By default, compute machines are deployed with a `soft-anti-affinity` server policy, but you can also choose `anti-affinity` or `soft-affinity` policies.

Web console

Developer perspective

- With this update, you can specify the name of a service binding connector in the **Topology** view while making a binding connection.
- With this update, creating pipelines workflow has now been enhanced:
  - You can now choose a user-defined pipeline from a drop-down list while importing your application from the **Import from Git** pipeline workflow.
  - Default webhooks are added for the pipelines that are created using **Import from Git** workflow and the URL is visible in the side panel of the selected resources in the **Topology** view.
  - You can now opt out of the default Tekton Hub integration by setting the parameter `enable-devconsole-integration` to `false` in the `TektonConfig` custom resource.

    Example `TektonConfig` CR to opt out of Tekton Hub integration

    ```
    ...
    hub:
      params:
        - name: enable-devconsole-integration
          value: 'false'
    ...
    ```

  - **Pipeline builder** contains the Tekton Hub tasks that are supported by the cluster, all other unsupported tasks are excluded from the list.
- With this update, the application export workflow now displays the export logs dialog or alert while the export is in progress. You can use the dialog to cancel or restart the exporting process.
- With this update, you can add your new Helm Chart Repository to the **Developer Catalog** by creating a custom resource. Refer to the quick start guides in the **Developer** perspective to add a new **ProjectHelmChartRepository**.
- With this update, you can now access community devfiles samples using the **Developer Catalog**.

Dynamic plug-in (Technology Preview)

Starting with OpenShift Container Platform 4.10, the ability to create OpenShift console dynamic plug-ins is now available as a Technology Preview feature. You can use this feature to customize your interface at runtime in many ways, including:

- Adding custom pages
- Adding perspectives and updating navigation items
- Adding tabs and actions to resource pages

For more information about the dynamic plug-in, see Adding a dynamic plug-in to the OpenShift

Container Platform web console.

Multicluster console (Technology Preview)

Starting with OpenShift Container Platform 4.10, the multicluster console is now available as a Technology Preview feature. By enabling this feature, you can connect to remote clusters' API servers from a single OpenShift Container Platform console.

Running a pod in debug mode

With this update, you can now view debug terminals in the web console. When a pod has a container that is in a `CrashLoopBackOff` state, a debug pod can be launched. A terminal interface is displayed and can be used to debug the crash looping container.

- This feature can be accessed by the pod status pop-up window, which is accessed by clicking on the status of a pod, provides links to debug terminals for each crash looping container within that pod.
- You can also access this feature on the **Logs** tab of the pod details page. A debug terminal link is displayed above the log window when a crash looping container is selected.

Additionally, the pod status pop-up window now provides links to the **Logs** and **Events** tabs of the pod details page.

Customized workload notifications

With this update, you can customize workload notifications on the **User Preferences** page. **User workload notifications** under the **Notifications** tab allows you to hide user workload notifications that appear on the **Cluster Overview** page or in your drawer.

Improved quota visibility

With this update, non-admin users are now able to view their usage of the `AppliedClusterResourceQuota` on the **Project Overview**, **ResourceQuotas**, and **API Explorer** pages to determine the cluster-scoped quota available for use. Additionally, `AppliedClusterResourceQuota` details can now be found on the **Search** page.

Cluster support level

OpenShift Container Platform now enables you to view support level information about your cluster on the **Overview** **Details** card, in the **Cluster Settings**, in the **About** modal, and adds a notification to your notifications drawer when your cluster is unsupported. From the **Overview** page, you can manage subscription settings under the **Service Level Agreement (SLA)**.

IBM Z and LinuxONE

With this release, IBM Z and LinuxONE are now compatible with OpenShift Container Platform 4.10. The installation can be performed with z/VM or RHEL KVM. For installation instructions, see the following documentation:

- Installing a cluster with z/VM on IBM Z and LinuxONE

- Installing a cluster with z/VM on IBM Z and LinuxONE in a restricted network
- Installing a cluster with RHEL KVM on IBM Z and LinuxONE
- Installing a cluster with RHEL KVM on IBM Z and LinuxONE in a restricted network

Notable enhancements

The following new features are supported on IBM Z and LinuxONE with OpenShift Container Platform 4.10:

- Horizontal pod autoscaling
- The following Multus CNI plug-ins are supported:
  - Bridge
  - Host-device
  - IPAM
  - IPVLAN
- Compliance Operator 0.1.49
- NMState Operator
- OVN-Kubernetes IPsec encryption
- Vertical Pod Autoscaler Operator

Supported features

The following features are also supported on IBM Z and LinuxONE:

- Currently, the following Operators are supported:
  - Cluster Logging Operator
  - Compliance Operator 0.1.49
  - Local Storage Operator
  - NFD Operator
  - NMState Operator
  - OpenShift Elasticsearch Operator
  - Service Binding Operator
  - Vertical Pod Autoscaler Operator
- Encrypting data stored in etcd
- Helm
- Multipathing
- Persistent storage using iSCSI
- Persistent storage using local volumes (Local Storage Operator)
- Persistent storage using hostPath

- Persistent storage using Fibre Channel
- Persistent storage using Raw Block
- OVN-Kubernetes
- Support for multiple network interfaces
- Three-node cluster support
- z/VM Emulated FBA devices on SCSI disks
- 4K FCP block device

These features are available only for OpenShift Container Platform on IBM Z and LinuxONE for 4.10:

- HyperPAV enabled on IBM Z and LinuxONE for the virtual machines for FICON attached ECKD storage

Restrictions

The following restrictions impact OpenShift Container Platform on IBM Z and LinuxONE:

- The following OpenShift Container Platform Technology Preview features are unsupported:
  - Precision Time Protocol (PTP) hardware
- The following OpenShift Container Platform features are unsupported:
  - Automatic repair of damaged machines with machine health checking
  - CodeReady Containers (CRC)
  - Controlling overcommit and managing container density on nodes
  - CSI volume cloning
  - CSI volume snapshots
  - FIPS cryptography
  - NVMe
  - OpenShift Metering
  - OpenShift Virtualization
  - Tang mode disk encryption during OpenShift Container Platform deployment
- Worker nodes must run Red Hat Enterprise Linux CoreOS (RHCOS)
- Persistent shared storage must be provisioned by using either OpenShift Data Foundation or other supported storage protocols
- Persistent non-shared storage must be provisioned using local storage, like iSCSI, FC, or using LSO with DASD, FCP, or EDEV/FBA

IBM Power

With this release, IBM Power is now compatible with OpenShift Container Platform 4.10. For installation instructions, see the following documentation:

- Installing a cluster on IBM Power
- Installing a cluster on IBM Power in a restricted network

Notable enhancements

The following new features are supported on IBM Power with OpenShift Container Platform 4.10:

- Horizontal pod autoscaling
- The following Multus CNI plug-ins are supported:
  - Bridge
  - Host-device
  - IPAM
  - IPVLAN
- Compliance Operator 0.1.49
- NMState Operator
- OVN-Kubernetes IPsec encryption
- Vertical Pod Autoscaler Operator

Supported features

The following features are also supported on IBM Power:

- Currently, the following Operators are supported:
  - Cluster Logging Operator
  - Compliance Operator 0.1.49
  - Local Storage Operator
  - NFD Operator
  - NMState Operator
  - OpenShift Elasticsearch Operator
  - SR-IOV Network Operator
  - Service Binding Operator
  - Vertical Pod Autoscaler Operator
- Encrypting data stored in etcd
- Helm
- Multipathing
- Multus SR-IOV
- NVMe
- OVN-Kubernetes

- Persistent storage using iSCSI
- Persistent storage using local volumes (Local Storage Operator)
- Persistent storage using hostPath
- Persistent storage using Fibre Channel
- Persistent storage using Raw Block
- Support for multiple network interfaces
- Support for Power10
- Three-node cluster support
- 4K Disk Support

Restrictions

The following restrictions impact OpenShift Container Platform on IBM Power:

- The following OpenShift Container Platform Technology Preview features are unsupported:
  - Precision Time Protocol (PTP) hardware
- The following OpenShift Container Platform features are unsupported:
  - Automatic repair of damaged machines with machine health checking
  - CodeReady Containers (CRC)
  - Controlling overcommit and managing container density on nodes
  - FIPS cryptography
  - OpenShift Metering
  - OpenShift Virtualization
  - Tang mode disk encryption during OpenShift Container Platform deployment
- Worker nodes must run Red Hat Enterprise Linux CoreOS (RHCOS)
- Persistent storage must be of the Filesystem type that uses local volumes, OpenShift Data Foundation, Network File System (NFS), or Container Storage Interface (CSI)

Security and compliance

Information regarding new features, enhancements, and bug fixes for security and compliance components can be found in the Compliance Operator and File Integrity Operator release notes.

For more information about security and compliance, see OpenShift Container Platform security and compliance.

Networking

Dual-stack services require that ipFamilyPolicy is specified

When you create a service that uses multiple IP address families, you must explicitly specify

`ipFamilyPolicy: PreferDualStack` or `ipFamilyPolicy: RequireDualStack` in your Service object definition. This change breaks backward compatibility with earlier releases of OpenShift Container Platform.

For more information, see BZ#2045576.

Change cluster network MTU after cluster installation

After cluster installation, if you are using the OpenShift SDN cluster network provider or the OVN-Kubernetes cluster network provider, you can change your hardware MTU and your cluster network MTU values. Changing the MTU across the cluster is disruptive and requires that each node is rebooted several times. For more information, see Changing the cluster network MTU.

Low-latency Redfish hardware event delivery (Developer Preview)

OpenShift Container Platform now provides a hardware event proxy that enables applications running on bare-metal clusters to respond quickly to Redfish hardware events, such as hardware changes and failures.

The hardware event proxy supports a publish-subscribe service that allows relevant applications to consume hardware events detected by Redfish. The proxy must be running on hardware that supports Redfish v1.8 and higher. An Operator manages the lifecycle of the `hw-event-proxy` container.

You can use a REST API to develop applications to consume and respond to events such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. Reliable end-to-end messaging without persistent stores is based on the Advanced Message Queuing Protocol (AMQP). The latency of the messaging service is in the 10 millisecond range.

This feature is supported for single node OpenShift clusters only.

OVN-Kubernetes support for gateway configuration

The OVN-Kubernetes CNI network provider adds support for configuring how egress traffic is sent to the node gateway. By default, egress traffic is processed in OVN to exit the cluster and traffic is not affected by specialized routes in the kernel routing table.

This enhancement adds a `gatewayConfig.routingViaHost` field. With this update, the field can be set at runtime as a post-installation activity and when it is set to `true`, egress traffic is sent from pods to the host networking stack. This update benefits highly specialized installations and applications that rely on manually configured routes in the kernel routing table.

This enhancement has an interaction with the Open vSwitch hardware offloading feature. With this update, when the `gatewayConfig.routingViaHost` field is set to `true`, you do not receive the performance benefits of the offloading because egress traffic is processed by the host networking stack.

For configuration information, see Configuration for the OVN-Kubernetes CNI cluster network provider.

Enhancements to networking metrics

The following metrics are now available for clusters. The metric names that start with `sdn_controller` are unique to the OpenShift SDN CNI network provider. The metric names that start with `ovn` are unique to the OVN-Kubernetes CNI network provider:

- `network_attachment_definition_instances{networks="egress-router"}`
- `openshift_unidle_events_total`
- `ovn_controller_bfd_run`
- `ovn_controller_ct_zone_commit`
- `ovn_controller_flow_generation`
- `ovn_controller_flow_installation`
- `ovn_controller_if_status_mgr`
- `ovn_controller_if_status_mgr_run`
- `ovn_controller_if_status_mgr_update`
- `ovn_controller_integration_bridge_openflow_total`
- `ovn_controller_ofctrl_seqno_run`
- `ovn_controller_patch_run`
- `ovn_controller_pinctrl_run`
- `ovnkube_master_ipsec_enabled`
- `ovnkube_master_num_egress_firewall_rules`
- `ovnkube_master_num_egress_firewalls`
- `ovnkube_master_num_egress_ips`
- `ovnkube_master_pod_first_seen_lsp_created_duration_seconds`
- `ovnkube_master_pod_lsp_created_port_binding_duration_seconds`
- `ovnkube_master_pod_port_binding_chassis_port_binding_up_duration_seconds`
- `ovnkube_master_pod_port_binding_port_binding_chassis_duration_seconds`
- `sdn_controller_num_egress_firewall_rules`
- `sdn_controller_num_egress_firewalls`
- `sdn_controller_num_egress_ips`

The `ovnkube_master_resource_update_total` metric is removed for the 4.10 release.

Switching between YAML view and a web console form

- Previously, changes were not retained when switching between **YAML view** and **Form view** on the web console. Additionally, after switching to **YAML view**, you could not return to **Form view**. With this update, you can now easily switch between **YAML view** and **Form view** on the web console without losing changes.

Listing pods targeted by network policies

When using the network policy functionality in the OpenShift Container Platform web console, the pods affected by a policy are listed. The list changes as the combined namespace and pod selectors in these policy sections are modified:

- Peer definition
- Rule definition
- Ingress
- Egress

The list of impacted pods includes only those pods accessible by the user.

Enhancement to must-gather to simplify network tracing

The `oc adm must-gather` command is enhanced in a way that simplifies collecting network packet captures.

Previously, `oc adm must-gather` could start a single debug pod only. With this enhancement, you can start a debug pod on multiple nodes at the same time.

You can use the enhancement to run packet captures on multiple nodes at the same time to simplify troubleshooting network communication issues. A new `--node-selector` argument provides a way to identify which nodes you are collecting packet captures for.

For more information, see Network trace methods and Collecting a host network trace.

Pod-level bonding for secondary networks (Technology Preview)

Bonding at the pod level is vital to enable workloads inside pods that require high availability and more throughput. With pod-level bonding, you can create a bond interface from multiple single root I/O virtualization (SR-IOV) virtual function interfaces in kernel mode interface. The SR-IOV virtual functions are passed into the pod and attached to a kernel driver.

Scenarios where pod-level bonding is required include creating a bond interface from multiple SR-IOV virtual functions on different physical functions. Creating a bond interface from two different physical functions on the host can be used to achieve high availability at pod level.

The current functionality of Bond CNI is available only in active-backup mode. For further details, see **BZ#2037214**.

Egress IP address support for clusters installed on public clouds

As a cluster administrator, you can associate one or more egress IP addresses with a namespace. An egress IP address ensures that a consistent source IP address is associated with traffic from a particular namespace that is leaving the cluster.

For the OVN-Kubernetes and OpenShift SDN cluster network providers, you can configure an egress IP address on the following public cloud providers:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure

To learn more, refer to the respective documentation for your cluster network provider:

Monitoring

OpenShift SDN cluster network provider network policy support for egress policies and ipBlock except

If you use the OpenShift SDN cluster network provider, you can now use egress rules in network policy with `ipBlock` and `ipBlock.except`. You define egress policies in the `egress` array of the `NetworkPolicy` object.

For more information, refer to About network policy.

Ingress Controller router compression

This enhancement adds the ability to configure global HTTP traffic compression on the HAProxy Ingress Controller for specific MIME types. This update enables gzip-compression of your ingress workloads when there are large amounts of compressible routed traffic.

For more information, see Using router compression.

Support for CoreDNS customization

A cluster administrator can now configure DNS servers to allow DNS name resolution through the configured servers for the default domain. A DNS forwarding configuration can have both the default servers specified in the `/etc/resolv.conf` file and the upstream DNS servers.

For more information, see Using DNS forwarding.

Support for configuring the maximum length of the syslog message in the Ingress Controller

You can now set the maximum length of the syslog message in the Ingress Controller to any value between 480 and 4096 bytes.

For more information, see Ingress Controller configuration parameters.

Set CoreDNS forwarding policy

You can now set the CoreDNS forwarding policy through the DNS Operator. The default value is `Random`, and you can also set the value to `RoundRobin` or `Sequential`.

For more information, see Using DNS forwarding.

Open vSwitch hardware offloading support for SR-IOV

You can now configure Open vSwitch hardware offloading to increase data processing performance on compatible bare metal nodes. Hardware offloading is a method for processing data that removes data processing tasks from the CPU and transfers them to the dedicated data processing unit of a network interface controller. Benefits of this feature include faster data processing, reduced CPU workloads, and lower computing costs.

For more information, see Configuring hardware offloading.

Creating DNS records by using the Red Hat External DNS Operator (Technology Preview)

You can now create DNS records by using the Red Hat External DNS Operator on cloud providers such as AWS, Azure, and GCP. You can install the External DNS Operator using OperatorHub. You can use parameters to configure `ExternalDNS` as required.

For more information, see Installing the External DNS Operator.

Mutable Ingress Controller endpoint publishing strategy enhancement

Cluster administrators can now configure the Ingress Controller endpoint publishing strategy to change the load-balancer scope between `Internal` and `External` in OpenShift Container Platform.

For more information, see Ingress Controller endpoint publishing strategy.

OVS hardware offloading for clusters on RHOSP (Technology Preview)

For clusters that run on Red Hat OpenStack Platform (RHOSP), you can enable Open vSwitch (OVS) hardware offloading.

For more information, see Enabling OVS hardware offloading.

Reduction of RHOSP resources created by Kuryr

For clusters that run on RHOSP, Kuryr now only creates Neutron networks and subnets for namespaces that have at least one pod on the pods network. Additionally, pools in a namespace are populated after at least one pod on the pods network is created in the namespace.

Support for RHOSP DCN (Technology Preview)

You can now deploy a cluster on a Red Hat OpenStack Platform (RHOSP) deployment that uses a distributed compute node (DCN) configuration. This deployment configuration has several limitations:

- Only RHOSP version 16 is supported.
- For RHOSP 16.1.4, only hyper-converged infrastructure (HCI) and Ceph technologies are supported at the edge.
- For RHOSP 16.2, non-HCI and Ceph technologies are supported as well.
- Networks must be created ahead of time (Bring Your Own Network) as either tenant or

provider networks. These networks must be scheduled in the appropriate availability zones.

Support for external cloud providers for clusters on RHOSP (Technology Preview)

Clusters that run on RHOSP can now use Cloud Provider OpenStack. This capability is available as part of the `TechPreviewNoUpgrade` feature set.

Configuring host network interfaces with NMState on installer-provisioned clusters

OpenShift Container Platform now provides a `networkConfig` configuration setting for installer-provisioned clusters, which takes an NMState YAML configuration to configure host interfaces. During installer-provisioned installations, you can add the `networkConfig` configuration setting and the NMState YAML configuration to the `install-config.yaml` file. Additionally, you can add the `networkConfig` configuration setting and the NMState YAML configuration to the bare metal host resource when using the Bare Metal Operator.

The most common use case for the `networkConfig` configuration setting is to set static IP addresses on a host's network interface during installation or while expanding the cluster.

For more information, see Configuring host network interfaces in the install-config.yaml file.

Boundary clock and PTP enhancements to linuxptp services

You can now specify multiple network interfaces in a `PtpConfig` profile to allow nodes running RAN vDU applications to serve as a Precision Time Protocol Telecom Boundary Clock (PTP T-BC). Interfaces configured as boundary clocks now also support PTP fast events.

For more information, see Configuring linuxptp services as boundary clock.

Support for Intel 800-Series Columbiaville NICs

Intel 800-Series Columbiaville NICs are now fully supported for interfaces configured as boundary clocks or ordinary clocks. Columbiaville NICs are supported in the following configurations:

- Ordinary clock
- Boundary clock synced to the Grandmaster clock
- Boundary clock with one port synchronizing from an upstream source clock, and three ports providing downstream timing to destination clocks

For more information, see Configuring PTP devices.

Kubernetes NMState Operator is GA for bare-metal, IBM Power, IBM Z, and LinuxONE installations

OpenShift Container Platform now provides the Kubernetes NMState Operator for bare-metal, IBM Power, IBM Z, and LinuxONE installations. The Kubernetes NMState Operator is still a Technology Preview for all other platforms. See About the Kubernetes NMState Operator for additional details.

Hardware

Enhancements to MetalLB load balancing

The following enhancements to MetalLB and the MetalLB Operator are included in this release:

- Support for Border Gateway Protocol (BGP) is added.
- Support for Bidirectional Forwarding Detection (BFD) in combination with BGP is added.
- Support for IPv6 and dual-stack networking is added.
- Support for specifying a node selector on the `speaker` pods is added. You can now control which nodes are used for advertising load balancer service IP addresses. This enhancement applies to layer 2 mode and BGP mode.
- Validating web hooks are added to ensure that address pool and BGP peer custom resources are valid.
- The `v1alpha1` API version for the `AddressPool` and `MetalLB` custom resource definitions that were introduced in the 4.9 release are deprecated. Both custom resources are updated to the `v1beta1` API version.
- Support for speaker pod tolerations in the MetalLB custom resource definition is added.

For more information, see About MetalLB and the MetalLB Operator.

Support for modifying host firmware settings

OpenShift Container Platform supports the `HostFirmwareSettings` and `FirmwareSchema` resources. When deploying OpenShift Container Platform on bare metal hosts, there are times when you need to make changes to the host either before or after provisioning. This can include inspecting the host's firmware and BIOS details. There are two new resources that you can use with the Bare Metal Operator (BMO):

- `HostFirmwareSettings`: You can use the `HostFirmwareSettings` resource to retrieve and manage the BIOS settings for a host. The resource contains the complete BIOS configuration returned from the baseboard management controller (BMC). Whereas, the firmware field in the `BareMetalHost` resource returns three vendor-independent fields, the `HostFirmwareSettings` resource typically comprises many BIOS settings of vendor-specific fields per host model.
- `FirmwareSchema`: You can use the `FirmwareSchema` to identify the host's modifiable BIOS values and limits when making changes to host firmware settings.

See Bare metal configuration for additional details.

Storage

Storage metrics indicator

- With this update, workloads can securely share `Secrets` and `ConfigMap` objects across namespaces using inline ephemeral `csi` volumes provided by the Shared Resource CSI Driver. Container Storage Interface (CSI) volumes and the Shared Resource CSI Driver are

Technology Preview features. (BUILD-293)

Console Storage Plug-in enhancement

- A new feature has been added to the Console Storage Plug-in that adds Aria labels throughout the installation flow for screen readers. This provides better accessibility for users that use screen readers to access the console.
- A new feature has been added that provides metrics indicating the amount of used space on volumes used for persistent volume claims (PVCs). This information appears in the PVC list, and in the PVC details in the **Used** column. (**BZ#1985965**)

Persistent storage using the Alibaba AliCloud Disk CSI Driver Operator

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for AliCloud Disk. The AliCloud Disk Driver Operator that manages this driver is generally available, and enabled by default in OpenShift Container Platform 4.10.

For more information, see AliCloud Disk CSI Driver Operator.

Persistent storage using the Microsoft Azure File CSI Driver Operator (Technology Preview)

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Azure File. The Azure File Driver Operator that manages this driver is in Technology Preview.

For more information, see Azure File CSI Driver Operator.

Persistent storage using the IBM VPC Block CSI Driver Operator

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for IBM Virtual Private Cloud (VPC) Block. The IBM VPC Block Driver Operator that manages this driver is generally available, and enabled by default in OpenShift Container Platform 4.10.

For more information, see IBM VPC Block CSI Driver Operator.

Persistent storage using VMware vSphere CSI Driver Operator is generally available

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for vSphere. This feature was previously introduced as a Technology Preview feature in OpenShift Container Platform 4.8 and is now generally available and enabled by default in OpenShift Container Platform 4.10.

For more information, see vSphere CSI Driver Operator.

vSphere CSI Driver Operator installation requires:

- Certain minimum component versions installed. See CSI driver installation on vSphere

clusters

- Removal of any non-Red Hat vSphere CSI driver (Removing a non-Red Hat vSphere CSI Operator Driver)
- Removal of any storage class named `thin-csi`

Clusters are still upgraded even if the preceding conditions are not met, but it is recommended that you meet these conditions to have a supported vSphere CSI Operator Driver.

Persistent storage using Microsoft Azure Disk CSI Driver Operator is generally available

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for Azure Disk. This feature was previously introduced as a Technology Preview feature in OpenShift Container Platform 4.8 and is now generally available, and enabled by default in OpenShift Container Platform 4.10.

For more information, see Azure Disk CSI Driver Operator.

Persistent storage using AWS Elastic File Storage CSI Driver Operator is generally available

OpenShift Container Platform is capable of provisioning persistent volumes (PVs) using the Container Storage Interface (CSI) driver for AWS Elastic File Storage (EFS). This feature was previously introduced as a Technology Preview feature in OpenShift Container Platform 4.9 and is now generally available in OpenShift Container Platform 4.10.

For more information, see AWS EFS CSI Driver Operator.

Automatic CSI migration supports Microsoft Azure file (Technology Preview)

Starting with OpenShift Container Platform 4.8, automatic migration for in-tree volume plug-ins to their equivalent Container Storage Interface (CSI) drivers became available as a Technology Preview feature. This feature now supports automatic migration for the Azure File in-tree plug-in to the Azure File CSI driver.

For more information, see CSI automatic migration.

Automatic CSI migration supports VMware vSphere (Technology Preview)

Starting with OpenShift Container Platform 4.8, automatic migration for in-tree volume plug-ins to their equivalent Container Storage Interface (CSI) drivers became available as a Technology Preview feature. This feature now supports automatic migration for the vSphere in-tree plug-in to the vSphere CSI driver.

For more information, see CSI automatic migration.

Using fsGroup to reduce pod timeouts

If a storage volume contains many files (~1,000,000 or greater), you may experience pod timeouts.

OpenShift Container Platform 4.10 introduces the ability to use `fsGroup` and `fsGroupChangePolicy` to skip recursive permission change for the storage volume, therefore helping to avoid pod timeout problems.

For more information, see Using fsGroup to reduce pod timeouts.

Registry

Operator lifecycle

Disabling copied CSVs to support large clusters

When an Operator is installed by Operator Lifecycle Manager (OLM), a simplified copy of its cluster service version (CSV) is created in every namespace that the Operator is configured to watch. These CSVs are known as copied CSVs; they identify controllers that are actively reconciling resource events in a given namespace.

On large clusters, with namespaces and installed Operators potentially in the hundreds or thousands, copied CSVs can consume an untenable amount of resources, such as OLM's memory usage, cluster etcd limits, and networking bandwidth. To support these larger clusters, cluster administrators can now disable copied CSVs for Operators that are installed with the `AllNamespaces` mode.

For more details, see Configuring Operator Lifecycle Manager features.

Generic and complex constraints for dependencies

Operators with specific dependency requirements can now use complex constraints or requirement expressions. The new `olm.constraint` bundle property holds dependency constraint information. A message field allows Operator authors to convey high-level details about why a particular constraint was used.

For more details, see Operator Lifecycle Manager dependency resolution.

Operator Lifecycle Manager support for Hypershift

Operator Lifecycle Manager (OLM) components, including Operator catalogs, can now run entirely on Hypershift-managed control planes. This capability does not incur any cost to tenants on worker nodes.

Operator Lifecycle Manager support for ARM

Previously, the default Operator catalogs did not support ARM. With this enhancement, Operator Lifecycle Manager (OLM) adds default Operator catalogs to ARM clusters. As a result, the OperatorHub now includes content by default for Operators that support ARM. (**BZ#1996928**)

Operator development

Hybrid Helm Operator (Technology Preview)

The standard Helm-based Operator support in the Operator SDK has limited functionality compared to the Go-based and Ansible-based Operator support that has reached the Auto Pilot capability (level V) in the Operator maturity model.

Starting in OpenShift Container Platform 4.10 as a Technology Preview feature, the Operator SDK includes the Hybrid Helm Operator to enhance the existing Helm-based support abilities through Go APIs. Operator authors can generate an Operator project beginning with a Helm chart, and then add advanced, event-based logic to the Helm reconciler in Go language. Authors can use Go to continue adding new APIs and custom resource definitions (CRDs) in the same project.

For more details, see Operator SDK tutorial for Hybrid Helm Operators.

Custom metrics for Ansible-based Operators

Operator authors can now use the Ansible-based Operator support in the Operator SDK to expose custom metrics, emit Kubernetes events, and provide better logging.

For more details, see Exposing custom metrics for Ansible-based Operators.

Object pruning for Go-based Operators

The `operator-lib` pruning utility lets Go-based Operators clean up objects, such as jobs or pods, that can stay in the cluster and use resources. The utility includes common pruning strategies for Go-based Operators. Operator authors can also use the utility to create custom hooks and strategies.

For more information about the pruning utility, see Object pruning utility for Go-based Operators.

Digest-based bundle for disconnected environments

With this enhancement, Operator SDK can now package an Operator project into a bundle that works in a disconnected environment with Operator Lifecycle Manager (OLM). Operator authors can run the `make bundle` command and set `USE_IMAGE_DIGESTS` to `true` to automatically update your Operator image reference to a digest rather than a tag. To use the command, you must use environment variables to replace hard-coded related image references.

For more information about developing Operators for disconnected environments, see Enabling your Operator for restricted network environments.

Builds

- With this update, you can use CSI volumes in OpenShift Builds, which is a Technology Preview feature. This feature relies on the newly introduced Shared Resource CSI Driver and the Insights Operator to import RHEL Simple Content Access (SCA) certificates. For example, by using this feature, you can run entitled builds with `SharedSecret` objects and install entitled RPM packages during builds rather than copying your RHEL subscription credentials

and certificates into the builds' namespaces. (BUILD-274)

The `SharedSecret` objects and OpenShift Shared Resources feature are only available if you enable the `TechPreviewNoUpgrade` feature set. These Technology Preview features are not part of the default features. Enabling this feature set cannot be undone and prevents upgrades. This feature set is not recommended on production clusters. See Enabling Technology Preview features using FeatureGates.

- With this update, workloads can securely share `Secrets` and `ConfigMap` objects across namespaces using inline ephemeral `csi` volumes provided by the Shared Resource CSI Driver. Container Storage Interface (CSI) volumes and the Shared Resource CSI Driver are Technology Preview features. (BUILD-293)

Jenkins

- With this update, you can run Jenkins agents as sidecar containers. You can use this capability to run any container image in a Jenkins pipeline that has a correctly configured pod template and Jenkins file. Now, to compile code, you can run two new pod templates named `java-build` and `nodejs-builder` as sidecar containers with Jenkins. These two pod templates use the latest Java and NodeJS versions provided by the `java` and `nodejs` image streams in the `openshift` namespace. The previous non-sidecar `maven` and `nodejs` pod templates have been deprecated. (JKNS-132)

Machine API

Azure Ephemeral OS disk support

With this enhancement, you can create a machine set running on Azure that deploys machines on Ephemeral OS disks. Ephemeral OS disks use local VM capacity rather than remote Azure Storage.

For more information, see Machine sets that deploy machines on Ephemeral OS disks.

Azure Accelerated Networking support

With this release, you can enable Accelerated Networking for Microsoft Azure VMs by using the Machine API. Accelerated Networking uses single root I/O virtualization (SR-IOV) to provide VMs with a more direct path to the switch.

For more information, see Accelerated Networking for Microsoft Azure VMs.

Global Azure availability set support

With this release, you can use availability sets in global Azure regions that do not have multiple availability zones to ensure high availability.

GPU support on Google Cloud Platform

Google Cloud Platform (GCP) Compute Engine enables users to add GPUs to VM instances. Workloads that benefit from access to GPU resources can perform better on compute machines with this feature enabled. With this release, you can define which supported GPU to use for an instance by using the Machine API.

For more information, see Enabling GPU support for a machine set.

Cluster autoscaler node utilization threshold

With this enhancement, you can specify a node utilization threshold in the `ClusterAutoscaler` resource definition. This threshold represents the node utilization level below which an unnecessary node is eligible for deletion.

For more information, see About the cluster autoscaler.

Machine Config Operator

Enhanced configuration drift detection

With this enhancement, the Machine Config Daemon (MCD) now checks nodes for configuration drift if a filesystem write event occurs for any of the files specified in the machine config and before a new machine config is applied, in addition to node bootup. Previously, the MCD checked for configuration drift only at node bootup. This change was made because node reboots do not occur frequently enough to avoid the problems caused by configuration drift until an administrator can correct the issue.

Configuration drift occurs when the on-disk state of a node differs from what is configured in the machine config. The Machine Config Operator (MCO) uses the MCD to check nodes for configuration drift and, if detected, sets that node and machine config pool (MCP) to `degraded`.

For more information about configuration drift, see Understanding configuration drift detection.

Nodes

Linux control groups version 2 (Developer Preview)

You can now enable Linux control groups version 2 (cgroups v2) on specific nodes in your cluster. The OpenShift Container Platform process for enabling cgroups v2 disables all cgroups version 1 controllers and hierarchies. The OpenShift Container Platform cgroups version 2 feature is in Developer Preview and is not supported by Red Hat at this time. For more information, see Enabling Linux control groups version 2 (cgroups v2).

Support for swap memory use on nodes (Technology Preview)

You can enable swap memory use for OpenShift Container Platform workloads on a per-node basis. For more information, see Enabling swap memory use on nodes.

Place nodes into maintenance mode by using the Node Maintenance Operator

The Node Maintenance Operator (NMO) cordons off nodes from the rest of the cluster and drains all the pods from the nodes. By placing nodes under maintenance, you can investigate problems with a machine, or perform operations on the underlying machine, that might result in a node failure. This is a standalone version of NMO. If you installed OpenShift Virtualization, then you must use the NMO that is bundled with it.

Node Health Check Operator enhancements (Technology Preview)

The Node Health Check Operator provides these new enhancements:

- Support for running in disconnected mode
- Prevent conflicts with machine health check. For more information, see About how node health checks prevent conflicts with machine health checks

Poison Pill Operator enhancements

The Poison Pill Operator uses `NodeDeletion` as its default remediation strategy. The `NodeDeletion` remediation strategy removes the `node` object.

In OpenShift Container Platform 4.10, the Poison Pill Operator introduces a new remediation strategy called `ResourceDeletion`. The `ResourceDeletion` remediation strategy removes the pods and associated volume attachments on the node rather than the `node` object. This strategy helps to recover workloads faster.

Control plane node migration on RHOSP

You can now migrate control plane nodes from one RHOSP host to another without encountering a service disruption.

Red Hat OpenShift Logging

In OpenShift Container Platform 4.7, *Cluster Logging* became *Red Hat OpenShift Logging*. For more information, see Release notes for Red Hat OpenShift Logging.

Monitoring

The monitoring stack for this release includes the following new and modified features.

Monitoring stack components and dependencies

Updates to versions of monitoring stack components and dependencies include the following:

- Alertmanager to 0.23.0
- Grafana to 8.3.4
- kube-state-metrics to 2.3.0
- node-exporter to 1.3.1

- prom-label-proxy to 0.4.0
- Prometheus to 2.32.1
- Prometheus adapter to 0.9.1
- Prometheus operator to 0.53.1
- Thanos to 0.23.1

New page for metrics targets in the OpenShift Container Platform web console

A new **Metrics Targets** page in the OpenShift Container Platform web console shows targets for default OpenShift Container Platform projects and for user-defined projects. You can use this page to view, search, and filter the endpoints that are currently targeted for scraping, which helps you to identify and troubleshoot problems.

Monitoring components updated to use TLS authentication for metrics collection

With this release, all monitoring components are now configured to use mutual TLS authentication, rather than Bearer Token static authentication for metrics collection. TLS authentication is more resilient to Kubernetes API outages and decreases the load on the Kubernetes API.

Cluster Monitoring Operator updated to use the global TLS security profile

With this release, the Cluster Monitoring Operator components now honor the global OpenShift Container Platform `tlsSecurityProfile` settings. The following components and services now use the TLS security profile:

- Alertmanager pods (ports 9092 and 9097)
- kube-state-metrics pod (ports 8443 and 9443)
- openshift-state-metrics pod (ports 8443 and 9443)
- node-exporter pods (port 9100)
- Grafana pod (port 3002)
- prometheus-adapter pods (port 6443)
- prometheus-k8s pods (ports 9092 and 10902)
- Thanos query pods (ports 9092, 9093 and 9094)
- Prometheus Operator (ports 8080 and 8443)
- telemeter-client pod (port 8443)

If you have enabled user-defined monitoring, the following pods now use the profile:

- prometheus-user-workload pods (ports 9091 and 10902)
- prometheus-operator pod (ports 8080 and 8443)

Changes to alerting rules

- **New**
  - Added a `namespace` label to all Thanos alerting rules.
  - Added the `openshift_io_alert_source="platform"` label to all platform alerts.
- **Changed**
  - Renamed `AggregatedAPIDown` to `KubeAggregatedAPIDown`.
  - Renamed `AggregatedAPIErrors` to `KubeAggregatedAPIErrors`.
  - Removed the `HighlyAvailableWorkloadIncorrectlySpread` alert.
  - Improved the description of the `KubeMemoryOvercommit` alert.
  - Improved `NodeFilesystemSpaceFillingUp` alerts to make it consistent with the Kubernetes garbage collection thresholds.
  - Excluded `ReadOnlyMany` volumes from the `KubePersistentVolumeFillingUp` alerts.
  - Extended `PrometheusOperator` alerts to include the Prometheus operator running in the `openshift-user-workload-monitoring` namespace.
  - Replaced the `ThanosSidecarPrometheusDown` and `ThanosSidecarUnhealthy` alerts by `ThanosSidecarNoConnectionToStartedPrometheus`.
  - Changed the severity of `KubeletTooManyPods` from `warning` to `info`.
  - Enabled exclusion of specific persistent volumes from `KubePersistentVolumeFillingUp` alerts by adding the `alerts.k8s.io/KubePersistentVolumeFillingUp: disabled` label to a persistent volume resource.

Red Hat does not guarantee backward compatibility for recording rules or alerting rules.

Changes to metrics

- Pod-centric cAdvisor metrics available at the slice level have been dropped.
- The following metrics are now exposed:
  - `kube_poddisruptionbudget_labels`
  - `kube_persistentvolumeclaim_labels`
  - `kube_persistentvolume_labels`
- Metrics with the name `kube_*annotation` have been removed from `kube-state-metrics`.

Red Hat does not guarantee backward compatibility for metrics.

Added hard anti-affinity rules and pod disruption budgets for certain components

With this release, hard anti-affinity rules and pod disruption budgets have been enabled for the following monitoring components to reduce downtime during patch upgrades:

- Alertmanager

As part of this change, the number of Alertmanager replicas has been reduced from three to two. However, the persistent volume claim (PVC) for the removed third replica is not automatically removed as part of the upgrade process. If you have configured persistent storage for Alertmanager, you can remove this PVC manually from the Cluster Monitoring Operator. See the "Known Issues" section for more information.

- Prometheus adapter
- Prometheus
- Thanos Querier

If you have enabled user-defined monitoring, the following components also use these rules and budgets:

- Prometheus
- Thanos Ruler

Alert routing for user-defined projects (Technology Preview)

This release introduces a Technology Preview feature in which an administrator can enable alert routing for user-defined projects monitoring. Users can then add and configure alert routing for their user-defined projects.

Alertmanager

Access to the third-party Alertmanager web user interface from the OpenShift Container Platform route has been removed.

Prometheus

- OpenShift Container Platform cluster administrators can now configure query logging for Prometheus.
- Access to the third-party Prometheus web user interface is deprecated and will be removed in a future OpenShift Container Platform release.

Prometheus adapter

- The Prometheus adapter now uses the Thanos Querier API rather than the Prometheus API.
- OpenShift Container Platform cluster administrators can now configure audit logs for the Prometheus adapter.

Thanos Querier

- Access to the third-party Thanos Querier web user interface from the OpenShift Container Platform route has been removed.

- The `/api/v1/labels`, `/api/v1/label/*/values`, and `/api/v1/series` endpoints on the Thanos Querier tenancy port are now exposed.
- OpenShift Container Platform cluster administrators can now configure query logging.
- If user workload monitoring is enabled, access to the third-party Thanos Ruler web user interface from the OpenShift Container Platform route has been removed.

Grafana

Access to the third-party Grafana web user interface is deprecated and will be removed in a future OpenShift Container Platform release.

Scalability and performance

New Special Resource Operator metrics

The Special Resource Operator (SRO) now exposes metrics to help you watch the health of your SRO custom resources and objects. For more information, see Prometheus Special Resource Operator metrics.

Special Resource Operator custom resource definition fields

Using `oc explain` for Special Resource Operator (SRO) now provides online documentation for SRO custom resource definitions (CRD). This enhancement provides better specifics for CRD fields. (**BZ#2031875**)

New Node Tuning Operator metric added to Telemetry

A Node Tuning Operator (NTO) metric is now added to Telemetry. Follow the procedure in Showing data collected by Telemetry to see all the metrics collected by Telemetry.

NFD Topology Updater is now available

The Node Feature Discovery (NFD) Topology Updater is a daemon responsible for examining allocated resources on a worker node. It accounts for resources that are available to be allocated to new pod on a per-zone basis, where a zone can be a Non-Uniform Memory Access (NUMA) node. See Using the NFD Topology Updater for more information.

Hyperthreading-aware CPU manager policy (Technology Preview)

Hyperthreading-aware CPU manager policy in OpenShift Container Platform is now available without the need for extra tuning. The cluster administrator can enable this feature if required. Hyperthreads are abstracted by the hardware as logical processors. Hyperthreading allows a single physical processor to execute two heavyweight threads (processes) at the same time, dynamically sharing processor resources.

Updating managed clusters with Cluster Group Upgrades Operator (Developer Preview)

You can now use the upstream Cluster Group Upgrades Operator to perform updates on multiple single-node Openshift clusters by using Red Hat Advanced Cluster Management

(RHACM) policies. For more information, see About the Cluster Group Upgrades Operator configuration.

Indication of done for ZTP

A new tool is available that simplifies the process of checking for a completed zero touch provisioning (ZTP) installation using the Red Hat Advanced Cluster Management (RHACM) static validator inform policy. It provides an Indication of done for ZTP installations by capturing the criteria for a completed installation and validating that it moves to a compliant state only when ZTP provisioning of the spoke cluster is complete.

This policy can be used for deployments of single node clusters, three-node clusters, and standard clusters. To learn more about the validator inform policy, see Indication of done

Enhancements to ZTP

For OpenShift Container Platform 4.10, there are a number of updates that make it easier to configure the hub cluster and generate source CRs. New PTP and UEFI secure boot features for spoke clusters are also available. A summary of these features is below:

- You can add or modify existing source CRs in the `ztp-site-generate` container, rebuild it, and make it available to the hub cluster, typically from the disconnected registry associated with the hub cluster.
- You can configure PTP fast events for vRAN clusters that are deployed using the GitOps zero touch provisioning (ZTP) pipeline.
- You can configure UEFI secure boot for vRAN clusters that are deployed using the GitOps ZTP pipeline.
- You can use Cluster Group Upgrades Operator to orchestrate the application of the configuration CRs to the hub cluster.

ZTP support for multicluster deployment

Zero touch provisioning (ZTP) provides support for multicluster deployment, including single node clusters, three-node clusters, and standard OpenShift clusters. This includes the installation of OpenShift and deployment of the distributed units (DUs) at scale. This gives you the ability to deploy nodes with master, worker, and master and worker roles. ZTP multinode support is implemented through the use of `SiteConfig` and `PolicyGenTemplate` custom resources (CRs). The overall flow is identical to the ZTP support for single node clusters, with some differentiation in configuration depending on the type of cluster:

In the `SiteConfig` file:

- Single node clusters must have exactly one entry in the `nodes` section.
- Three-node clusters must have exactly three entries defined in the `nodes` section.
- Standard OpenShift clusters must have exactly three entries in the `nodes` section with role: master and one or more additional entries with role: worker.

The `PolicyGenTemplate` file tells the Policy Generator where to categorize the generated policies.

Example `PolicyGenTemplate` files provide you with example files to simplify your deployments:

- The example common `PolicyGenTemplate` file is common across all types of clusters.
- Example group `PolicyGenTemplate` files for single node, three-node, and standard clusters are provided.
- Site-specific `PolicyGenTemplate` files specific to each site are provided.

To learn more about multicluster deployment, see Deploying a site

Backup and restore

Developer experience

Pruning deployment replica sets (Technology Preview)

This release introduces a Technology Preview flag `--replica-sets` to the `oc adm prune deployments` command. By default, only replication controllers are pruned with the `oc adm prune deployments` command. When you set `--replica-sets` to `true`, replica sets are also included in the pruning process.

For more information, see Pruning deployment resources.

Insights Operator

Importing simple content access certificates

In OpenShift Container Platform 4.10, Insights Operator now imports your simple content access certificates from Red Hat OpenShift Cluster Manager by default.

For more information, see Importing simple content access certificates with Insights Operator.

Insights Operator data collection enhancements

To reduce the amount of data sent to Red Hat, Insights Operator only gathers information when certain conditions are met. For example, Insights Operator only gathers the Alertmanager logs when Alertmanager fails to send alert notifications.

In OpenShift Container Platform 4.10, the Insights Operator collects the following additional information:

- (Conditional) The logs from pods where the `KubePodCrashlooping` and `KubePodNotReady` alerts are firing
- (Conditional) The Alertmanager logs when the `AlertmanagerClusterFailedToSendAlerts` or `AlertmanagerFailedToSendAlerts` alerts are firing
- Silenced alerts from Alertmanager
- The node logs from the journal unit (kubelet)
- The `CostManagementMetricsConfig` from clusters with `costmanagement-metrics-operator`

installed

- The time series database status from the monitoring stack Prometheus instance
- Additional information about the OpenShift Container Platform scheduler

With this additional information, Red Hat improves OpenShift Container Platform functionality and enhances Insights Advisor recommendations.

Authentication and authorization

Syncing group membership from OpenID Connect identity providers

This release introduces support for synchronizing group membership from an OpenID Connect provider to OpenShift Container Platform upon user login. You can enable this by configuring the `groups` claim in the OpenShift Container Platform OpenID Connect identity provider configuration.

For more information, see Sample OpenID Connect CRs.

Additional supported OIDC providers

The Okta and Ping Identity OpenID Connect (OIDC) providers are now tested and supported with OpenShift Container Platform.

For the full list of OIDC providers, see Supported OIDC providers.

oc commands now obtain credentials from Podman configuration locations

Previously, `oc` commands that used the registry configuration, for example `oc registry login` or `oc image` commands, obtained credentials from Docker configuration locations. With OpenShift Container Platform 4.10, if a registry entry cannot be found in the default Docker configuration location, `oc` commands obtain the credentials from Podman configuration locations. You can set your preference to either `docker` or `podman` by using the `REGISTRY_AUTH_PREFERENCE` environment variable to prioritize the location.

Users also have the option to use the `REGISTRY_AUTH_FILE` environment variable, which serves as an alternative to the existing `--registry-config` CLI flag. The `REGISTRY_AUTH_FILE` environment variable is also compatible with `podman`.

Support for Google Cloud Platform Workload Identity

You can now use the Cloud Credential Operator (CCO) utility `ccoctl` to configure the CCO to use the Google Cloud Platform Workload Identity. When the CCO is configured to use GCP Workload Identity, the in-cluster components can impersonate IAM service accounts using short-term, limited-privilege security credentials to components.

For more information, see Using manual mode with GCP Workload Identity.

In OpenShift Container Platform 4.10.8, image registry support for using GCP Workload Identity was removed due to the discovery of an adverse impact to the image registry. To use the image registry on an OpenShift Container Platform 4.10.8 cluster that uses Workload Identity, you must configure the image registry to use long-lived credentials instead. This mitigation is planned to last until Workload Identity support for the image registry is restored in a later release.

For more information, see Image registry support for Google Cloud Platform Workload Identity removed.

Notable technical changes

OpenShift Container Platform 4.10 introduces the following notable technical changes.

TLS X.509 certificates must have a Subject Alternative Name

X.509 certificates must have a properly set the Subject Alternative Name field. If you update your cluster without this, you risk breaking your cluster or rendering it inaccessible.

In older versions of OpenShift Container Platform, X.509 certificates worked without a Subject Alternative Name, so long as the Common Name field was set. This behavior was removed in OpenShift Container Platform 4.6.

In some cases, certificates without a Subject Alternative Name continued to work in OpenShift Container Platform 4.6, 4.7, 4.8, and 4.9. Because it uses Kubernetes 1.23, OpenShift Container Platform 4.10 does not allow this under any circumstances.

Cloud controller managers for additional cloud providers

The Kubernetes community plans to deprecate the Kubernetes controller manager in favor of using cloud controller managers to interact with underlying cloud platforms. As a result, there is no plan to add Kubernetes controller manager support for any new cloud platforms. The implementation that is added in this release of OpenShift Container Platform supports using cloud controller managers for Google Cloud Platform (GCP), VMware vSphere, IBM Cloud, and Alibaba Cloud as a Technology Preview.

To learn more about the cloud controller manager, see the Kubernetes Cloud Controller Manager documentation.

To manage the cloud controller manager and cloud node manager deployments and lifecycles, use the Cluster Cloud Controller Manager Operator.

For more information, see the Cluster Cloud Controller Manager Operator entry in the *Platform Operators reference*.

Operator SDK v1.16.0

OpenShift Container Platform 4.10 supports Operator SDK v1.16.0. See Installing the Operator

SDK CLI to install or update to this latest version.

Operator SDK v1.16.0 supports Kubernetes 1.22.

Many deprecated `v1beta1` APIs were removed in Kubernetes 1.22, including `sigs.k8s.io/controller-runtime v0.10.0` and `controller-gen v0.7`. This is a breaking change if you need to scaffold `v1beta1` APIs for custom resource definitions (CRDs) or webhooks to publish your project into older cluster versions.

For more information about changes introduced in Kubernetes 1.22, see Validating bundle manifests for APIs removed from Kubernetes 1.22 and Beta APIs removed from Kubernetes 1.22 in the OpenShift Container Platform 4.9 release notes.

If you have any Operator projects that were previously created or maintained with Operator SDK v1.10.1, see Upgrading projects for newer Operator SDK versions to ensure your projects are upgraded to maintain compatibility with Operator SDK v1.16.0.

Changed Cluster Autoscaler alert severity

Previously, the `ClusterAutoscalerUnschedulablePods` alert showed a severity of `warning`, which suggested it required developer intervention. This alert is informational and does not describe a problematic condition that requires intervention. With this release, the `ClusterAutoscalerUnschedulablePods` alert is reduced in severity from `warning` to `info`. (**BZ#2025230**)

Deprecated and removed features

Some features available in previous releases have been deprecated or removed.

Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments. For the most recent list of major functionality deprecated and removed within OpenShift Container Platform 4.10, refer to the table below. Additional details for more fine-grained functionality that has been deprecated and removed are listed after the table.

In the table, features are marked with the following statuses:

- **GA**: *General Availability*
- **DEP**: *Deprecated*
- **REM**: *Removed*

Table 1. Deprecated and removed features tracker

Feature	OCP 4.8	OCP 4.9	OCP 4.10
Package manifest format (Operator Framework)	REM	REM	REM
SQLite database format for Operator catalogs	GA	DEP	DEP

Feature	OCP 4.8	OCP 4.9	OCP 4.10
`oc adm catalog build`	REM	REM	REM
`--filter-by-os` flag for `oc adm catalog mirror`	REM	REM	REM
v1beta1 CRDs	DEP	REM	REM
Docker Registry v1 API	DEP	REM	REM
Metering Operator	DEP	REM	REM
Scheduler policy	DEP	DEP	REM
`ImageChangesInProgress` condition for Cluster Samples Operator	DEP	DEP	DEP
`MigrationInProgress` condition for Cluster Samples Operator	DEP	DEP	DEP
Use of `v1` without a group in `apiVersion` for OpenShift Container Platform resources	DEP	REM	REM
Use of `dhclient` in RHCOS	DEP	REM	REM
Cluster Loader	DEP	DEP	REM
Bring your own RHEL 7 compute machines	DEP	DEP	REM
`lastTriggeredImageID` field in the `BuildConfig` spec for Builds	DEP	REM	REM
Jenkins Operator	DEP	DEP	REM
HPA custom metrics adapter based on Prometheus	REM	REM	REM
vSphere 6.7 Update 2 or earlier	GA	DEP	DEP
Virtual hardware version 13	GA	DEP	DEP
VMware ESXi 6.7 Update 3 or earlier	GA	DEP	DEP
Minting credentials for Microsoft Azure clusters	GA	GA	REM
Persistent storage using FlexVolume			DEP
Non-sidecar pod templates for Jenkins			DEP
Support for Google Cloud Platform Workload Identity	-	-	REM

Deprecated features

IBM POWER8, IBM z13 all models, LinuxONE Emperor, LinuxONE Rockhopper, and x86_64 v1 architectures will be deprecated

RHCOS functionality in IBM POWER8, IBM z13 all models, LinuxONE Emperor, LinuxONE Rockhopper, and AMD64 (x86_64) v1 CPU architectures will be deprecated in an upcoming release. Additional details for when support will discontinue for these architectures will be

announced in a future release.

AMD and Intel 64-bit architectures (x86-64-v2) will still be supported.

Default Docker configuration location deprecation

Previously, `oc` commands that used a registry configuration would obtain credentials from the Docker configuration location, which was `~/.docker/config.json` by default. This has been deprecated and will be replaced by a Podman configuration location in a future version of OpenShift Container Platform.

Empty file and stdout support deprecation in oc registry login

Support for empty files using the `--registry-config` and `--to` flags in `oc registry login` has been deprecated. Support for `-` (standard output) has also been deprecated as an argument when using `oc registry login`. They will be removed in a future version of OpenShift Container Platform.

Non-sidecar pod templates for Jenkins deprecation

In OpenShift Container Platform 4.10, the non-sidecar `maven` and `nodejs` pod templates for Jenkins are deprecated. These pod templates are planned for removal in a future release. Bug fixes and support are provided through the end of that future life cycle, after which no new feature enhancements are made. Instead, with this update, you can run Jenkins agents as sidecar containers. (JKNS-257)

Third-party monitoring components user interface deprecation

For the following monitoring stack components, access to third-party web user interfaces (UIs) is deprecated and is planned to be removed in a future OpenShift Container Platform release:

- Grafana
- Prometheus

As an alternative, users can navigate to the **Observe** section of the OpenShift Container Platform web console to access dashboards and other UIs for platform components.

Persistent storage using FlexVolume

In OpenShift Container Platform 4.10, persistent storage using FlexVolume is deprecated. This feature is still fully supported, but only important bugs will be fixed. However, it may be removed in a future OpenShift Container Platform release. Out-of-tree Container Storage Interface (CSI) driver is the recommended way to write volume drivers in OpenShift Container Platform. Maintainers of FlexVolume drivers should implement a CSI driver and move users of FlexVolume to CSI. Users of FlexVolume should move their workloads to CSI driver.

Removed features

OpenShift Container Platform 4.10 removes the Jenkins Operator, which was a Technology

Preview feature, from the **OperatorHub** page in the OpenShift Container Platform web console interface. Bug fixes and support are no longer available.

Instead, you can continue to deploy Jenkins on OpenShift Container Platform by using the templates provided by the Samples Operator. Alternatively, you can install the Jenkins Helm Chart from the Developer Catalog by using the **Helm** page in the **Developer** perspective of the web console.

OpenShift CLI (oc) commands removed

The following OpenShift CLI (`oc`) commands were removed with this release:

- `oc adm completion`
- `oc adm config`
- `oc adm options`

Scheduler policy removed

Support for configuring a scheduler policy has been removed with this release. Use a scheduler profile instead to control how pods are scheduled onto nodes.

RHEL 7 support for compute machines removed

Support for running Red Hat Enterprise Linux (RHEL) 7 compute machines in OpenShift Container Platform has been removed. If you prefer using RHEL compute machines, they must run on RHEL 8.

You cannot upgrade RHEL 7 compute machines to RHEL 8. You must deploy new RHEL 8 hosts, and the old RHEL 7 hosts must be removed.

Third-party monitoring component user interface access removed

With this release, you can no longer access third-party web user interfaces (UIs) for the following monitoring stack components:

- Alertmanager
- Thanos Querier
- Thanos Ruler (if user workload monitoring is enabled)

Instead, you can navigate to the **Observe** section of the OpenShift Container Platform web console to access metrics, alerting, and metrics targets UIs for platform components.

Support for minting credentials for Microsoft Azure removed

Support for using the Cloud Credential Operator (CCO) in mint mode on Microsoft Azure clusters has been removed. This change is due to the planned retirement of the Azure AD Graph API by Microsoft on 30 June 2022 and is being backported to all supported versions of OpenShift Container Platform in z-stream updates.

For previously installed Azure clusters that use mint mode, the CCO attempts to update existing secrets. If a secret contains the credentials of previously minted app registration service principals, it is updated with the contents of the secret in `kube-system/azure-credentials`. This behavior is similar to passthrough mode.

For clusters with the credentials mode set to its default value of `""`, the updated CCO automatically changes from operating in mint mode to operating in passthrough mode. If your cluster has the credentials mode explicitly set to mint mode (`"Mint"`), you must change the value to `""` or `"Passthrough"`.

In addition to the `Contributor` role that is required by mint mode, the modified app registration service principals now require the `User Access Administrator` role that is used for passthrough mode.

While the Azure AD Graph API is still available, the CCO in upgraded versions of OpenShift Container Platform attempts to clean up previously minted app registration service principals. Upgrading your cluster before the Azure AD Graph API is retired might avoid the need to clean up resources manually.

If the cluster is upgraded to a version of OpenShift Container Platform that no longer supports mint mode after the Azure AD Graph API is retired, the CCO sets an `OrphanedCloudResource` condition on the associated `CredentialsRequest` but does not treat the error as fatal. The condition includes a message similar to `unable to clean up App Registration / Service Principal: <app_registration_name>`. Cleanup after the Azure AD Graph API is retired requires manual intervention using the Azure CLI tool or the Azure web console to remove any remaining app registration service principals.

To clean up resources manually, you must find and delete the affected resources.

1. Using the Azure CLI tool, filter the app registration service principals that use the `<app_registration_name>` from an `OrphanedCloudResource` condition message by running the following command:

```
$ az ad app list --filter "displayname eq '<app_registration_name>'" --query
'[].objectId'
```

Example output

```
[
  "038c2538-7c40-49f5-abe5-f59c59c29244"
]
```

2. Delete the app registration service principal by running the following command:

```
$ az ad app delete --id 038c2538-7c40-49f5-abe5-f59c59c29244
```

After cleaning up resources manually, the `OrphanedCloudResource` condition persists because the CCO cannot verify that the resources were cleaned up.

Image registry support for Google Cloud Platform Workload Identity removed

OpenShift Container Platform 4.10.3 introduced the ability to configure the Cloud Credential Operator (CCO) to use Google Cloud Platform (GCP) Workload Identity by using the CCO utility `ccoctl`.

In OpenShift Container Platform 4.10.8, image registry support for using GCP Workload Identity is removed due to the discovery of an adverse impact to the image registry. To use the image registry on an OpenShift Container Platform 4.10.8 cluster that uses Workload Identity, you must configure the image registry to use long-lived credentials instead. This mitigation is planned to last until Workload Identity support for the image registry is restored in a later release.

Prerequisites

- You have installed an OpenShift Container Platform cluster that uses manual mode with GCP Workload Identity.
- You have obtained and initialized the gcloud CLI.

Procedure

1. Obtain the email address of the IAM service account that the `ccoctl` created for the OpenShift Container Platform image registry by running the following command:

   ```
   $ gcloud iam service-accounts list --filter="displayName=<name>-openshift-
   image-registry-gcs"
   ```

   where `<name>` is the user-defined name for all `ccoctl`-created GCP resources used for tracking.

2. Create a long-lived credentials key file for the OpenShift Container Platform image registry by running the following command:

   ```
   $ gcloud iam service-accounts keys create
   <path_to_image_registry_service_account_keyfile> --iam-account
   =<image_registry_service_account_email>
   ```

   where:

   - `<path_to_image_registry_service_account_keyfile>` is the path to a location you choose in which to store the key file for the image registry's IAM service account.
   - `<image_registry_service_account_email>` is the email address of the IAM service account that the `ccoctl` created for the OpenShift Container Platform image registry.

3. Generate a Base64-encoded string for the image registry key file by running the following command:

```
$ cat <path_to_image_registry_service_account_keyfile> | base64 -w 0
```

4. In the `installer-cloud-credentials` secret in the `openshift-image-registry` namespace, replace the contents of the `data.service_account.json` field with the Base64-encoded key file by running the following command:

```
$ oc patch secret installer-cloud-credentials -n openshift-image-registry
--patch '{"data":{"service_account.json":"<base64_encoded_credentials_key>"}}'
```

5. Continue the upgrade process.

Bug fixes

Bare Metal Hardware Provisioning

- Previously, using a MAC address to configure a provisioning network interface was unsupported when switching the provisioning network from `Disabled` to `Managed`. With this update, a `provisioningMacAddresses` field is added to the `provisioning.metal3.io` CRD. Use this field to identify the provisioning network interface using its MAC address rather than its name. (**BZ#2000081**)
- Previously, Ironic failed to attach virtual media images for provisioning SuperMicro X11/X12 servers because these models expect a non-standard device string, for example `UsbCd`, for CD-based virtual media. With this update, provisioning now overrides `UsbCd` on SuperMicro machines provisioned with CD-based virtual media. (**BZ#2009555**)
- Previously, Ironic failed to attach virtual media images on SuperMicro X11/X12 servers due to overly restrictive URl validations on the BMCs of these machines. With this update, the `filename` parameter has now been removed from the URL if the virtual media image is backed by a local file. As a result, the parameter still passes if the image is backed by an object store. (**BZ#2011626**)
- Previously, the `curl` utility, used by the machine downloader image, did not support classless inter-domain routing (CIDR) with `no_proxy`. A a result, any CIDR in`noProxy` was ignored when downloading the Red Hat Enterprise Linux CoreOS (RHCOS) image. With this update, proxies are now removed from the environment before calling `curl` when appropriate. As a result, when downloading the machine image, any CIDR in `no_proxy` is no longer ignored. (**BZ#1990556**)
- Previously, virtual media based deployments of OpenShift Container Platform have been observed to intermittently fail on iDRAC hardware types. This occurred when outstanding Lifecycle Controller jobs clashed with virtual media configuration requests. With this update, virtual media deployment failure has been fixed by purging any Lifecycle Controller job while registering iDRAC hardware prior to deployment. (**BZ#1988879**)
- Previously, users had to enter a long form of an IPv6 address in the installation configuration file, for example `2001:0db8:85a3:0000:0000:8a2e:0370:7334`. Ironic could not find an interface matching this IP address causing the installation to fail. With this update, the IPv6 address supplied by the user is converted to a short form address, for example,

`2001:db8:85a3::8a2e:370:7334`. As a result, installation is now successful. (**BZ#2010698**)

Builds

- Before this update, if you created a build configuration containing an image change trigger in OpenShift Container Platform 4.7.x or earlier, the image change trigger might trigger builds continuously.

  This issue happened because, with the deprecation and removal of the `lastTriggeredImageID` field from the `BuildConfig` spec for Builds, the image change trigger controller stopped checking that field before instantiating builds. OpenShift Container Platform 4.8 introduced new fields in the status that the image change trigger controller needed to check, but did not.

  With this update, the image change trigger controller continuously checks the correct fields in the spec and status for the last triggered image ID. Now, it only triggers a build when necessary. (BZ#2004203)

- Before this update, image references in Builds needed to specify the Red Hat registry name explicitly. With this update, if an image reference does not contain the registry, the Build searches the Red Hat registries and other well-known registries to locate the image. (BZ#2011293)

Jenkins

- Before this update, version 1.0.48 of the OpenShift Jenkins Sync Plugin introduced a `NullPointerException` error when Jenkins notified the plug-in of new jobs that were not associated with an OpenShift Jenkins Pipeline Build Strategy Build Config. Ultimately, this error was benign because there was no `BuildConfig` object to associate with the incoming Jenkins Job. Core Jenkins ignored the exception in our plug-in and moved on to the next listener. However, a long stack trace showed up in the Jenkins log that distracted users. With this update, the plug-in resolves the issue by making the proper checks to avoid this error and the subsequent stack trace. (BZ#2030692)
- Before this update, performance improvements in version 1.0.48 of the OpenShift Sync Jenkins plug-in incorrectly specified the labels accepted for `ConfigMap` and `ImageStream` objects intended to map into the Jenkins Kubernetes plug-in pod templates. As a result, the plug-in no longer imported pod templates from `ConfigMap` and `ImageStream` objects with a `jenkins-agent` label.

  This update corrects the accepted label specification so that the plug-in imports pod templates from `ConfigMap` and `ImageStream` objects that have the `jenkins-agent` label. (2034839)

Cloud Compute

- Previously, editing a machine specification on Red Hat OpenStack Platform (RHOSP) would cause OpenShift Container Platform to attempt to delete and recreate the machine. As a result, this caused an unrecoverable loss of the node it was hosting. With this fix, any edits made to the machine specification after creation are ignored. (**BZ#1962066**)

- Previously, on clusters that run on Red Hat OpenStack Platform (RHOSP), floating IP addresses were not reported for machine objects. As a result, certificate signing requests (CSRs) that the kubelet created were not accepted, preventing nodes from joining the cluster. All IP addresses are now reported for machine objects. (**BZ#2022627**)
- Previously, the check to ensure that the AWS machine was not updated before requeueing was removed. Consequently, problems arose when the AWS machine's virtual machine had been removed, but its object was still available. If this happens, the AWS machine would requeue in an infinite loop and could not be deleted or updated. This update restores the check that was used to ensure that the AWS machine was not updated before requeueing. As a result, machines no longer requeue if they have been updated. (**BZ#2007802**)
- Previously, modifying a selector changed the list of machines that a machine set observed. As a result, leaks could occur because the machine set lost track of machines it had already created. This update ensures that the selector is immutable once created. As a result, machine sets now lists the correct machines. (**BZ#2005052**)
- Previously, if a virtual machine template had snapshots, an incorrect disk size was picked due to an incorrect usage of the `linkedClone` operation. With this update, the default clone operation is changed to `fullClone` for all situations. `linkedClone` must now be specified by the user. (**BZ#2001008**)
- Previously, the custom resource definition (CRD) schema requirements did not allow numeric values. Consequently, marshaling errors occurred during upgrades. This update corrects the schema requirements to allow both string and numeric values. As a result, marshaling errors are no longer reported by the API server conversion. (**BZ#1999425**)
- Previously, if the Machine API Operator was moved, or the pods were deployed as a result of a name change, the `MachineNotYetDeleted` metric would reset for each monitored machine. This update changes the metric query to ignore the source pod label. As a result, the `MachineNotYetDeleted` metric now properly alerts in scenarios where the Machine API Operator pod has been renamed. (**BZ#1986237**)
- Previously, egress IPs on vSphere were picked up by the vSphere cloud provider within the kubelet. These were unexpected by the certificate signing requests (CSR) approver. Consequently, nodes with egress IPs would not have their CSR renewals approved. This update allows the CSR approver to account for egress IPs. As a result, nodes with egress IPs on vSphere SDN clusters now continue to function and have valid CSR renewals. (**BZ#1860774**)
- Previously, worker nodes failed to start, and the installation program failed to generate URL images due to the broken path defaulting for the disk image and incompatible changes in the Google Cloud Platform (GCP) SDK. As a result, the machine controller was unable to create machines. This fix repairs the URL images by changing the base path in the GCP SDK. (**BZ#2009111**)
- Previously, the machine would freeze during the deletion process due to a lag in the vCenter's `powerOff` task. VMware showed the machine to be powered off, but OpenShift Container Platform reported it to be powered on, which resulted in the machine freezing during the deletion process. This update improves the `powerOff` task handling on vSphere to be checked before the task to delete from the database is created, which prevents the machine from freezing during the deletion process. (**BZ#2011668**)

- After installing or updating OpenShift Container Platform, the value of the metrics showed one pending CSR after the last CSR was reconciled. This resulted in the metrics reporting one pending CSR when there should be no pending CSRs. This fix ensures the pending CSR count is always valid post-approval by updating the metrics at the end of each reconcile loop. (**BZ#2013528**)
- Previously, AWS checked for credentials when the `cloud-provider` flag was set to empty string. The credentials were checked by making calls to the metadata service, even on non-AWS platforms. This caused latency in the ECR provider startup and AWS credential errors logged in all platforms, including non-AWS. This fix prevents the credentials check from making any requests to the metadata service to ensure that credential errors are no longer being logged. (**BZ#2015515**)
- Previously, the Machine API sometimes reconciled a machine before AWS had communicated VM creation across its API. As a result, AWS reported the VM does not exist and the Machine API considered it failed. With this release, the Machine API waits until the AWS API has synched before trying to mark the machine as provisioned. (**BZ#2025767**)
- Previously, a large volume of nodes created simultaneously on UPI clusters could lead to a large number of CSRs being generated. As a result, certificate renewals were not automated because the approver stops approving certificates when there are over 100 pending certificate requests. With this release, existing nodes are accounted for when calculating the approval cut-off and UPI clusters can now benefit from automated certificate renewal even with large scale refresh requests. (**BZ#2028019**)
- Previously, the generated list of instance types embedded in Machine API controllers was out of date. Some of these instance types were unknown and could not be annotated for scale-from-zero requirements. With this release, the generated list is updated to include support for newer instance types. (**BZ#2040376**)
- Previously, AWS Machine API controllers did not set the IOPS value for block devices other than the IO1 type, causing IOPS fields for GP3 block devices to be ignored. With this release, the IOPS is set on all supported block device types and users can set IOPS for block devices that are attached to the machine. (**BZ#2040504**)

Cloud Credential Operator

- Previously, when using the Cloud Credential Operator in manual mode on an Azure cluster, the `Upgradeable` status was not set to `False`. This behavior was different for other platforms. With this release, Azure clusters using the Cloud Credential Operator in manual mode have the `Upgradeable` status set to `False`. (**BZ#1976674**)
- Previously, the now unnecessary `controller-manager-service` service resource that was created by the Cloud Credential Operator was still present. With this release, the Cluster Version Operator cleans it up. (**BZ#1977319**)
- Previously, changes to the log level setting for the Cloud Credential Operator in the `CredentialsRequest` custom resource were ignored. With this release, logging verbosity can be controlled by editing the `CredentialsRequest` custom resource. (**BZ#1991770**)
- Previously, the Cloud Credential Operator (CCO) pod restarted with a continuous error when AWS was the default secret annotator for Red Hat OpenStack Platform (RHOSP). This update fixes the default setting for the CCO pod and prevents the CCO pod from failing.

(BZ#1996624)

Cluster Version Operator

- Previously, a pod might fail to start due to an invalid mount request that was not a part of the manifest. With this update, the Cluster Version Operator (CVO) removes any volumes and volume mounts from in-cluster resources that are not included in the manifest. This allows pods to start successfully. (BZ#2002834)
- Previously, when monitoring certificates were rotated, the Cluster Version Operator (CVO) would log errors and monitoring would be unable to query metrics until the CVO pod was manually restarted. With this update, the CVO monitors the certificate files and automatically recreates the metrics connection whenever the certificate files change. (BZ#2027342)

Console Storage Plug-in

- Previously, a loading prompt was not present while the persistent volumes (PVs) were being provisioned and the capacity was `0` TiB which created a confusing scenario. With this update, a loader is added for the loading state which provides details to the user if the PVs are still being provisioned or capacity is to be determined. It will also inform the user of any errors in the process. (BZ#1928285)
- Previously, the grammar was not correct in certain places and there were instances where translators were unable to interpret the context. This had a negative impact on readability. With this update, the grammar in various places is corrected, the storage classes for translators is itemized, and the overall readability is improved. (BZ#1961391)
- Previously, when pressing a pool inside the block pools page, the final `Ready` phase persisted after deletion. Consequently, the pool was in the `Ready` state even after deletion. This update redirects users to the **Pools** page and refreshes the pools after detention. (BZ#1981396)

Domain Name System (DNS)

- Previously, the DNS Operator did not cache responses from upstream resolvers that were configured using `spec.servers`. With this update, the DNS Operator now caches responses from all upstream servers. (BZ#2006803)
- Previously, the DNS Operator did not enable the `prometheus` pug-in in the server blocks for custom upstream resolvers. Consequently, CoreDNS did not report metrics for upstream resolvers and only reported metrics for the default server block. With this update, the DNS Operator was changed to enable the `prometheus` plug-in in all server blocks. CoreDNS now reports Prometheus metrics for custom upstream resolvers. (BZ#2020489)
- Previously, an upstream DNS that provided a response greater than 512 characters caused an application to fail. This occurred because it could not clone the repository from GitHub because to DNS could not be resolved. With this update, the `bufsize` for KNI CoreDNS is set to 521 to avoid name resolutions from GitHub. (BZ#1991067)
- When the DNS Operator reconciles its operands, the Operator gets the cluster DNS service object from the API to determine whether the Operator needs to create or update the service. If the service already exists, the Operator compares it with what the Operator

expects to get to determine whether an update is needed. Kubernetes 1.22, on which OpenShift Container Platform 4.9 is based, introduced a new `spec.internalTrafficPolicy` API field for services. The Operator leaves this field empty when it creates the service, but the API sets a default value for this field. The Operator was observing this default value and trying to update the field back to the empty value. This caused the Operator's update logic to continuously try to revert the default value that the API set for the service's internal traffic policy. With this fix, when comparing services to determine whether an update is required, the Operator now treats the empty value and default value for `spec.internalTrafficPolicy` as equal. As a result, the Operator no longer spuriously tries to update the cluster DNS service when the API sets a default value for the service's `spec.internalTrafficPolicy field`. (**BZ#2002461**)

- Previously, the DNS Operator did not enable the cache plug-in for server blocks in the CoreDNS `Corefile` configuration map corresponding to entries in the `spec.servers` field of the `dnses.operator.openshift.io/default` object. As a result, CoreDNS did not cache responses from upstream resolvers that were configured using `spec.servers`. With this bug fix, the DNS Operator is changed to enable the cache plug-in for all server blocks, using the same parameters that the Operator already configured for the default server block. CoreDNS now caches responses from all upstream resolvers. (**BZ#2006803**)

Image Registry

- Previously, the registry internally resolved `\docker.io` references into `\registry-1.docker.io` and used it to store credentials. As a result, credentials for `\docker.io` images could not be located. With this update, the `\registry-1.docker.io` hostname has been changed back to `\docker.io` when searching for credentials. As a result, the registry can correctly find credentials for `\docker.io images`. (**BZ#2024859**)
- Previously, the image pruner job did not retry upon failure. As a result, a single failure could degrade the Image Registry Operator until the next time it ran. With this fix, the temporary problems with the pruner do not degrade the Image Registry Operator. (**BZ#2051692**)
- Previously, the Image Registry Operator was modifying objects from the informer. As a result, these objects could be concurrently modified by the informer and cause race conditions. With this fix, controllers and informers have different copies of the object and do not have race conditions. (**BZ#2028030**)
- Previously, `TestAWSFinalizerDeleteS3Bucket` would fail because of an issue with the location of the configuration object in the Image Registry Operator. This update ensures that the configuration object is stored in the correct location. As a result, the Image Registry Operator no longer panics when running `TestAWSFinalizerDeleteS3Bucket`. (**BZ#2048443**)
- Previously, error handling caused the `access denied` error to be output as `authentication required`. This bug resulted in incorrect error logs. Through Docker distribution error handling, the error output was changed from `authentication required` to `access denied`. Now the `access denied` error provides more precise error logs. (**BZ#1902456**)
- Previously, the registry was immediately exiting on a shut down request. As a result, the router did not have time to discover that the registry pod was gone and could send requests to it. With this fix, when the pod is deleted it stays active for a few extra seconds to give other components time to discover its deletion. Now, the router does not send requests to non-existent pods during upgrades, which no longer leads to disruptions. (**BZ#1972827**)

- Previously, the registry proxied response from the first available mirrored registry. When a mirror registry was available but did not have the requested data, pull-through did not try to use other mirrors even if they contained the required data. With this fix, pull-through tries other mirror registries if the first mirror replied with `Not Found`. Now, pull-through can discover data if it exists on any mirror registry. (**BZ#2008539**)

Image Streams

- Previously, the image policy admission plug-in did not recognize deployment configurations, notably that stateful sets could be updated. As a result, image stream references stayed unresolved in deployment configurations when the `resolve-names` annotation was used. Now, the plug-in is updated so that it resolves annotations in deployment configurations and stateful sets. As a result, image stream tags get resolved in created and edited deployment configurations. (**BZ#2000216**)
- Previously, when global pull secrets were updated, existing API server pod pull secrets were not updated. Now, the mount point for the pull secret is changed from the `/var/lib/kubelet/config.json` file to the `/var/lib/kubelet` directory. As a result, the updated pull secret now appears in existing API server pods. (**BZ#1984592**)
- Previously, the image admission plug-in did not check annotations inside deployment configuration templates. As a result, annotations inside deployment configuration templates could not be handled in replica controllers, and they were ignored. Now, the image admission plug-in analyzes the template of deployment configurations. With this fix, the image admission plug-in recognizes the annotations on the deployment configurations and on their templates. (**BZ#2032589**)

Installer

- The OpenShift Container Platform Baremetal IPI installer previously used the first nodes defined under hosts in `install-config` as control plane nodes rather than filtering for the hosts with the `master` role. The role of `master` and `worker` nodes is now recognized when defined. (**BZ#2003113**)
- Before this update, it was possible to set host bits in the provisioning network CIDR. This could cause the provisioning IP to differ from what was expected leading to conflict with other IP addresses on the provisioning network. With this update, validation ensures that the provisioning network CIDR cannot contain host bits. If a provisioning Network CIDR includes host bits, the installation program stops and logs an error message. (**BZ#2006291**)
- Previously, pre-flight checks did not account for Red Hat OpenStack Platform (RHOSP) resource utilization. As a result, those checks failed with an incorrect error message when utilization, rather than quota, impeded installation. Pre-flight checks now process both RHOSP quota and utilization. The checks fail with correct error messages if the quota is sufficient but resources are not. (**BZ#2001317**)
- Before this update, the oVirt Driver could specify ReadOnlyMany (ROX) and ReadWriteMany (RWX) access modes when creating a PVC from a configuration file. This caused an error because the driver does not support shared disks and, as a result, could not support these access modes. With this update, the access mode has been limited to single node access. The system prevents any attempt to specify ROX or RWX when creating PVC and logs an

error message. (BZ#1882983)

- Previously, disk uploads in the Terraform provider were not handled properly. As a result, the OpenShift Container Platform installation program failed. With this update, disk upload handling has been fixed, and disk uploads succeed. (BZ#1917893)
- Previously, when installing a Microsoft Azure cluster with a special size, the installation program would check if the total number of virtual CPUs (vCPU) met the minimum resource requirement to deploy the cluster. Consequently, this could cause an install error. This update changes the check the installation program makes from the total number of vCPUs to the number of vCPUs available. As a result, a concise error message is given that lets the Operator know that the virtual machine size does not meet the minimum resource requirements. (BZ#2025788)
- Previously, RAM validation for Red Hat OpenStack Platform (RHOSP) checked for values using a wrong unit, and as a result the validation accepted flavors that did not meet minimum RAM requirements. With this fix, RAM validation now rejects flavors with insufficient RAM. (BZ#2009699)
- Previously, OpenShift Container Platform control plane nodes were missing Ingress security group rules when they were schedulable and deployed on Red Hat OpenStack Platform (RHOSP). As a result, OpenShift Container Platform deployments on RHOSP failed for compact clusters with no dedicated workers. This fix adds Ingress security group rules on Red Hat OpenStack Platform (RHOSP) when control plane nodes are schedulable. Now, you can deploy compact three-node clusters on RHOSP. (BZ#1955544)
- Previously, if you specified an invalid AWS region, the installation program continued to try and validate availability zones. This caused the installation program to become unresponsive for 60 minutes before timing out. The installation program now verifies the AWS region and service endpoints before availability zones, which reduces the amount of time the installation program takes to report the error. (BZ#2019977)
- Previously, you could not install a cluster to VMware vSphere if the vCenter hostname began with a number. The installation program has been updated and no longer treats this type of hostname as invalid. Now, a cluster deploys successfully when the vCenter hostname begins with a number. (BZ#2021607)
- Previously, if you specified a custom disk instance type when deploying a cluster on Microsoft Azure, the cluster might not deploy. This occurred because the installation program incorrectly determined that the minimum resource requirements had been met. The installation program has been updated, and now reports an error when the number of vcpus available for the instance type in the selected region does not meet the minimum resource requirements. (BZ#2025788)
- Previously, if you defined custom IAM roles when deploying an AWS cluster, you might have to manually remove bootstrap instance profiles after uninstalling the cluster. Intermittently, the installation program did not remove bootstrap instance profiles. The installation program has been updated, and all machine instance profiles are removed when the cluster is uninstalled. (BZ#2028695)
- Previously, the default provisioningIP value was different when the host bits were set in the provisioning network CIDR. This resulted in a different value for the provisioningIP than expected. This difference caused a conflict with the other IP addresses on the provisioning

network. This fix adds a validation to ensure that the ProvisioningNetworkCIDR does not have host bits set. As a result, if the ProvisioningNetworkCIDR has the host bits set, the installation program will stop and report the validation error. (BZ#2006291)

- Previously, the BMC driver IPMI was not supported for a secure UEFI boot. This resulted in an unsuccessful boot. This fix adds a validation check to ensure that `UEFISecureBoot` mode is not used with bare-metal drivers. As a result, a secure UEFI boot is successful. (BZ#2011893)
- With this update, the 4.8 UPI template is updated from version 3.1.0 to 3.2.0 to match the Ignition version. (BZ#1949672)
- Previously, when asked to mirror the contents of the base registry, the OpenShift Container Platform installation program would exit with a validation error, citing incorrect `install-config` file values for `imageContentSources`. With this update, the installation program now allows `imageContentSources` values to specify base registry names and the installation program no longer exits when specifying a base registry name. (BZ#1960378)
- Previously, the UPI ARM templates were attaching an SSH key to the virtual machine (VM) instances created. As a result, the creation of the VMs fails when the SSH key provided by the user is the `ed25519` type. With this update, the creation of the VMs succeeds regardless of the type of the SSH key provided by the user. (BZ#1968364)
- After successfully creating a `aws_vpc_dhcp_options_association` resource, AWS might still report that the resource does not exist. Consequently, the AWS Terraform provider fails the installation. With this update, you can retry the query of the `aws_vpc_dhcp_options_association` resource for a period of time after creation until AWS reports that the resource exists. As a result, installations succeed despite AWS reporting that the `aws_vpc_dhcp_options_association` resource does not exist. (BZ#2032521)
- Previously, when installing OpenShift Container Platform on AWS with local zones enabled, the installation program could create some resources on a local zone rather than an availability zone. This caused the installation program to fail because load balancers cannot run on local zones. With this fix, the installation program ignores local zones and only considers availability zones when installing cluster components. (BZ#1997059)
- Previously, terraform could attempt to upload the bootstrap ignition configuration file to Azure before it had finished creating the configuration file. If the upload started before the local file was created, the installation would fail. With this fix, terraform uploads the ignition configuration file directly to Azure rather than creating a local copy first. (BZ#2004313)
- Previously, a race condition could occur if the `cluster-bootstrap` and Cluster Version Operator components attempted to write a manifest for the same resource to the Kubernetes API at the same time. This could result in the `Authentication` resource being overwritten by a default copy, which removed any customizations made to that resource. With this fix, the Cluster Version Operator has been blocked from overwriting the manifests that come from the installation program. This prevents any user-specified customizations to the `Authentication` resource from being overwritten. (BZ#2008119)
- Previously, when installing OpenShift Container Platform on AWS, the installation program created the bootstrap machine using the `m5.large` instance type. This caused the installation to fail in regions where that instance type is not available. With this fix, the bootstrap machine uses the same instance type as the control plane machines. (BZ#2016955)
- Previously, when installing OpenShift Container Platform on AWS, the installation program

did not recognize EC2 G and Intel Virtualization Technology (VT) instances and defaulted them to X instances. This caused incorrect instance quotas to be applied to these instances. With this fix, the installation program recognizes EC2 G and VT instances and applies the correct instance quotas. (**BZ#2017874**)

Kubernetes API server

Kubernetes Scheduler

- Before this update, upgrading to the current release did not set the correct weights for the `TaintandToleration`, `NodeAffinity`, and `InterPodAffinity` parameters. This update resolves the issue so that upgrading correctly sets the weights for `TaintandToleration` to `3`, `NodeAffinity` to `2`, and `InterPodAffinity` to `2`. (**BZ#2039414**)
- In OpenShift Container Platform 4.10, code for serving insecure metrics is removed from the `kube-scheduler` code base. Now, metrics are served only through a secure server. Bug fixes and support are provided through the end of a future life cycle. After which, no new feature enhancements are made. (**BZ#1889488**)

Machine Config Operator

- Previously, the Machine Config Operator (MCO) stored pending configuration changes to the disk before operating system (OS) changes were applied. As a result, in situations such as power loss, the MCO assumed OS changes had already been applied on restart, and validation skipped over changes such as `kargs` and `kernel-rt`. With this update, configuration changes to disk are stored after OS changes are applied. Now, if power is lost during the configuration application, the MCO knows it must reattempt the configuration application on restart. (**BZ#1916169**)
- Previously, an old version of the Kubernetes client library in the `baremetal-runtimecfg` project prevented the timely closing of client connections following a VIP failover. This could result in long delays for monitor containers that rely on the API. This update allows the timely closing of client connections following a VIP failover. (**BZ#1995021**)
- Previously, when updating SSH keys, the Machine Config Operator (MCO) changed the owner and group of the `authorized_keys` file to `root`. This update ensures that the MCO preserves `core` as the owner and group when it updates the `authorized_keys` file. (BZ#1956739)
- Previously, a warning message sent by the `clone_slave_connection` function was incorrectly stored in a `new_uuid` variable, which is intended to store only the connection's UUID. As a result, `nmcli` commands that include the `new_uuid` variable were failing due to the incorrect value being stored in the `new_uuid` variable. With this fix, the `clone_slave_connection` function warning message is redirected to `stderr`. Now, `nmcli` commands that reference the `new_uuid` variable do not fail. (**BZ#2022646**)
- Previously, an old version of the Kubernetes client library was present in the `baremetal-runtimecfg` project. When a Virtual IP (VIP) failed, the client connections might not be closed in a timely manner. This could result in long delays for monitor containers that rely on talking to the API. This fix updates the client library. Now, connections are closed as expected on VIP failovers and the monitor containers do not hang for an excessive period of

time. (BZ#1995021)

- Before this update, the Machine Config Operator (MCO) stored pending configuration changes to disk before it applied them to Red Hat Enterprise Linux CoreOS (RHCOS). If a power loss interrupted the MCO from applying the configuration, it treated the configuration as applied and did not validate the changes. If this configuration contained invalid changes, applying them failed. With this update, the MCO saves a configuration to disk only after being applied. This way, if the power is lost while the MCO is applying the configuration, it reapplies the configuration after it restarts. (BZ#1916169)
- Before this update, when you used the Machine Config Operator (MCO) to create or update an SSH key, it set the owner and group of the `authorized_keys` file to `root`. This update resolves the issue. When the MCO creates or updates the `authorized_keys` file, it correctly sets or preserves `core` as the owner and group of the file. (BZ#1956739)
- Previously, in clusters that use Stateless Address AutoConfiguration (SLAAC), the Ironic `addr-gen-mode` parameter was not being persisted to the OVNKubernetes bridge. As a result, the IPv6 addresses could change when the bridge was created. This broke the cluster because node IP changes are unsupported. This fix persists the `addr-gen-mode` parameter when creating the bridge. The IP address is now consistent throughout the deployment process. (BZ#1990625)
- Previously, if a machine config included a compressed file with the `spec.config.storage.files.contents.compression` parameter set to `gzip`, the Machine Config Daemon (MCD) incorrectly wrote the compressed file to disk without extracting it. With this fix, the MCD now extracts a compressed file when the compression parameter is set to `gzip`. (BZ#1970218)
- Previously, `systemd` units were cleaned up only when completely removed. As a result, `systemd` units could not be unmasked by using a machine config because the masks were not being removed unless the `systemd` unit was completely removed. With this fix, when you configure a `systemd` unit as `mask: true` in a machine config, any existing masks are removed. As a result, `systemd` units can now be unmasked. (BZ#1966445)

Management Console

- Previously, the **OperatorHub** category and card links did not include valid `href` attributes. Consequently, the **OperatorHub** category and card links could not be opened in a new tab. This update adds valid `href` attributes to the **OperatorHub** category and card links. As a result, the **OperatorHub** and its card links can be opened in new tabs. (BZ#2013127)
- Previously, on the **Operand Details** page, a special case was created where the conditions table for the `status.conditions` property always rendered before all other tables. Consequently, the `status.conditions` table did not follow the ordering rules of descriptors, which caused unexpected behavior when users tried to change the order of the tables. This update removes the special case for `status.conditions` and only defaults to rendering it first if no descriptor is defined for that property. As a result, the table for `status.condition` is rendered according to descriptor ordering rules when a descriptor is defined on that property. (BZ#2014488)
- Previously, the **Resource Details** page metrics tab was exceeding the cluster-scoped Thanos endpoint. Consequently, users without authorization for this endpoint would receive a `401`

response for all queries. With this update, the Thanos tenancy endpoints are updated, and redundant namespace query arguments are removed. As a result, users with the correct role-based access control (RBAC) permissions can now see data in the metrics tab of the **Resource Details** page. (**BZ#2015806**)

- Previously, when an Operator added an API to an existing API group, it did not trigger API discovery. Consequently, new APIs were not seen by the front end until the page was refreshed. This update makes APIs added by Operators viewable by the front end without a page refresh. (**BZ#1815189**)
- Previously, in the Red Hat OpenShift Cluster Manager for Red Hat OpenStack Platform (RHOSP), the control plane was not translated into simplified Chinese. As a result, naming differed from OpenShift Container Platform documentation. This update fixes the translation issue in the Red Hat OpenShift Cluster Manager. (**BZ#1982063**)
- Previously, filtering of virtual tables in the Red Hat OpenShift Cluster Manager was broken. Consequently, all of the available `nodes` would not appear following a search. This update includes new virtual table logic that fixes the filtering issue in the Red Hat OpenShift Cluster Manager. (**BZ#1990255**)

Monitoring

- Previously, during OpenShift Container Platform upgrades, the Prometheus service could become unavailable because either two Prometheus pods were located on the same node or the two nodes running the pods rebooted during the same interval. This situation was possible because the Prometheus pods had soft anti-affinity rules regarding node placement and no `PodDisruptionBudget` resources provisioned. Consequently, metrics were not collected and rules were not evaluated over a period of time.

  To fix this issue, the Cluster Monitoring Operator (CMO) now configures hard anti-affinity rules to ensure that the two Prometheus pods are scheduled on different nodes. The CMO also provisions `PodDisruptionBudget` resources to ensure that at least one Prometheus pod is always running. As a result, during upgrades, the nodes now reboot in sequence to ensure that at least one Prometheus pod is always running. (**BZ#1933847**)

- Previously, the Thanos Ruler service would become unavailable when the node that contains the two Thanos Ruler pods experienced an outage. This situation occurred because the Thanos Ruler pods had only soft anti-affinity rules regarding node placement. Consequently, user-defined rules would not be evaluated until the node came back online.

  With this release, the Cluster Monitoring Operator (CMO) now configures hard anti-affinity rules to ensure that the two Thanos Ruler pods are scheduled on different nodes. As a result, a single-node outage no longer creates a gap in user-defined rule evaluation. (**BZ#1955490**)

- Previously, the Prometheus service would become unavailable when the two Prometheus pods were located on the same node and that node experienced an outage. This situation occurred because the Prometheus pods had only soft anti-affinity rules regarding node placement. Consequently, metrics would not be collected, and rules would not be evaluated until the node came back online.

With this release, the Cluster Monitoring Operator configures hard anti-affinity rules to ensure that the two Prometheus pods are scheduled on different nodes. As a result, Prometheus pods are now scheduled on different nodes, and a single node outage no longer creates a gap in monitoring.(**BZ#1949262**)

- Previously, during OpenShift Container Platform patch upgrades, the Alertmanager service might become unavailable because either the three Alertmanager pods were located on the same node or the nodes running the Alertmanager pods happened to reboot at the same time. This situation was possible because the Alertmanager pods had soft anti-affinity rules regarding node placement and no `PodDisruptionBudget` provisioned. This release enables hard anti-affinity rules and `PodDisruptionBudget` resources to ensure no downtime during patch upgrades for the Alertmanager and other monitoring components. (**BZ#1955489**)
- Previously, a false positive `NodeFilesystemSpaceFillingUp` alert was triggered when the file system space was occupied by many Docker images. For this release, the threshold to fire the `NodeFilesystemSpaceFillingUp` warning alert is now reduced to 20% space available, rather than 40%, which stops the false positive alert from firing. (**BZ#1987263**)
- Previously, alerts for the Prometheus Operator component did not apply to the Prometheus Operator that runs the `openshift-user-workload-monitoring` namespace when user-defined monitoring is enabled. Consequently, no alerts fired when the Prometheus Operator that manages the `openshift-user-workload-monitoring` namespace encountered issues.

  With this release, alerts have been modified to monitor both the `openshift-monitoring` and `openshift-user-workload-monitoring` namespaces. As a result, cluster administrators receive alert notifications when the Prometheus Operator that manages user-defined monitoring encounters issues. (**BZ#2001566**)

- Previously, if the number of DaemonSet pods for the `node-exporter` agent was not equal to the number of nodes in the cluster, the Cluster Monitoring Operator (CMO) would report a condition of `degraded`. This issue would occur when one of the nodes was not in the `ready` condition.

  This release now verifies that the number of DaemonSet pods for the `node-exporter` agent is not less than the number of ready nodes in the cluster. This process ensures that a `node-exporter` pod is running on every active node. As a result, the CMO will not report a degraded condition if one of the nodes is not in a ready state. (**BZ#2004051**)

- This release fixes an issue in which some pods in the monitoring stack would start before TLS certificate-related resources were present, which resulted in failures and restarts. (**BZ#2016352**)
- Previously, if reporting metrics failed due to reaching the configured sample limit, the metrics target would still appear with a status of `Up` in the web console UI even though the metrics were missing. With this release, Prometheus bypasses the sample limit setting for reporting metrics, and the metrics now appear regardless of the sample limit setting. (**BZ#2034192**)

Networking

- When using the OVN-Kubernetes network provider in OpenShift Container Platform versions

prior to 4.8, the node routing table was used for routing decisions. In newer versions of OpenShift Container Platform, the host routing table is bypassed. In this release, you can now specify whether you want to use or bypass the host kernel networking stack for traffic routing decisions. (**BZ#1996108**)

- Previously, when Kuryr was used in a restricted installation with proxy, the Cluster Network Operator was not enforcing usage of the proxy to allow communication with the Red Hat OpenStack Platform (RHOSP) API. Consequently, cluster installation did not progress. With this update, the Cluster Network Operator can communicate with the RHOSP API through the proxy. As a result, installation now succeeds. (**BZ#1985486**)
- Before this update, the SRIOV webhook blocked the creation of network policies on OpenShift Container Platform on Red Hat OpenStack Platform (RHOSP) environments. With this update, the SRIOV webhook reads and validates the RHOSP metadata and can now be used to create network policies. (**BZ#2016334**)
- Previously, the `MachineConfig` object could not be updated because the SRIOV Operator did not pause the `MachineConfig` pool object. With this update, the SRIOV Operator pauses the relevant machine config pool before running any configuration requiring reboot. (**BZ#2021151**)
- Previously, there were timing issues with `keepalived` that resulted in its termination when it should have been running. This update prevents multiple `keepalived` commands from being sent in a short period of time. As a result, timing issues are no longer a problem and `keepalive` continuously runs. (**BZ#2022050**)
- Previously, when Kuryr was used in a restricted installation with proxy, the Cluster Networking Operator was not enforcing usage of the proxy to allow communication with the Red Hat OpenStack Platform (RHOSP) API. Consequently, cluster installation did not progress. With this update, the Cluster Network Operator can communicate with the RHOSP API through the proxy. As a result, installation now succeeds. (**BZ#1985486**)
- Previously, pods that used secondary interfaces with IP addresses provided by the Whereabouts Container Network Interface (CNI) plug-in might get stuck in the `ContainerCreating` state because of IP address exhaustion. Now, Whereabouts properly accounts for released IP addresses from cluster events, such as reboots, that previously were not tracked. (BZ#1914053)
- Previously, when using the OpenShift SDN cluster network provider, idled services used an increasing amount of CPU to un-idle services. In this release, the idling code for kube-proxy is optimized to reduce CPU utilizing for service idling. (**BZ#1966521**)
- Previously, when using the OVN-Kubernetes cluster network provider, the presence of any unknown field in an internal configuration map could cause the OVN-Kubernetes pods to fail to start during a cluster upgrade. Now the presence of unknown fields causes a warning, rather than a failure. As a result, the OVN-Kubernetes pods now successfully start during a cluster upgrade. (**BZ#1988440**)
- Previously, a webhook for the SR-IOV Network Operator blocked network policies for OpenShift installations on OpenStack. Users were not able to create SR-IOV network policies. This update fixes the webhook. Users can now create SR-IOV network policies for installations on OpenStack. (**BZ#2016334**)
- Previously, the CRI-O runtime engine passed pod UIDs by using the `K8S_POD_UID` variable. But

when pods were deleted and recreated at the same time that Multus was setting up networking for the deleted pod's sandbox, this method resulted in additional metadata and unnecessary processing. In this update, Multus handles pod UIDs, and unnecessary metadata processing is avoided. (**BZ#2017882**)

- Previously, in deployments of OpenShift on a single node, default settings for the SR-IOV Network Operator prevented users from making certain modifications to nodes. By default, after configuration changes are applied, affected nodes are drained and then restarted with the new configuration. This behavior does not work when there is only one node. In this update, when you install the SR-IOV Network Operator in a single-node deployment, the Operator changes its configuration so the `.spec.disableDrain` field is set to `true`. Users can now apply configuration changes successfully in single-node deployments. (**BZ#2021151**)
- Client-go versions 1.20 and earlier did not have sufficient technique for retrying requests to the Kubernetes API. As a result, retries to the Kubernetes API were not sufficient. This update fixes the problem by introducing client-go 1.22. (**BZ#2052062**)

Node

- Previously, network, IPC, and UTS namespace resources managed by CRI-O were only freed when the Kubelet removed stopped pods. With this update, the Kubelet frees these resources when the pods are stopped. (**BZ#2003193**)
- Previously, when logging into a worker node, messages might appear indicating a `systemd-coredump` services failure. This was due to the unnecessary inclusion of the `system-systemd` namespace for containers. A filter now prevents this namespace from impacting the workflow. (**BZ#1978528**)
- Previously, when clusters were restarted, the status of terminated pods might have been reset to `Running`, which would result in an error. This has been corrected and now all terminated pods remain terminated and active pods reflect their correct status. (**BZ#1997478**)
- Previously, certain stop signals were ignored in OpenShift Container Platform, causing services in the container to continue running. With an update to the signal parsing library, all stop signals are now respected. (**BZ#2000877**)
- Previously, pod namespaces managed by CRI-O, for example network, IPC, and UTS, were not unmounted when the pod was removed. This resulted in leakage, driving the Open vSwitch CPU to 100%, which caused pod latency and other performance impacts. This has been resolved and pod namespaces are unmounted when removed. (**BZ#2003193**)

OpenShift CLI (oc)

- Previously, due to the increasing number of custom resource definitions (CRD) installed in the cluster, the requests reaching for API discovery were limited by client code restrictions. Now, both the limit number and QPS have been boosted, and client-side throttling should appear less frequently. (**BZ#2042059**)
- Previously, some minor requests did not have the user agent string set correctly, so the default Go user agent string was used instead for `oc`. The user agent string is now set correctly for all mirror requests, and the expected `oc` user agent string is now sent to registries. (**BZ#1987257**)

- Previously, `oc debug` assumed that it was always targeting Linux-based containers by trying to run a Bash shell, and if Bash was not present in the container, it attempted to debug as a Windows container. The `oc debug` command now uses pod selectors to determine the operating system of the containers and now works properly on both Linux and Windows-based containers. (**BZ#1990014**)
- Previously, the `--dry-run` flag was not working properly for several `oc set` subcommands, so `--dry-run=server` was performing updates to resources rather than performing a dry run. The `--dry-run` flags are now working properly to perform dry runs on the `oc set` subcommands. (**BZ#2035393**)

OpenShift containers

- Previously, a container using SELinux could not read `/var/log/containers` log files due to a missing policy. This update makes all log files in `/var/log` accessible including those accessed through symlink. (**BZ#2005997**)

OpenShift Controller Manager

- Previously, the `openshift_apps_deploymentconfigs_last_failed_rollout_time` metric improperly set the `namespace` label as the value of the `exported_namespace` label. The `openshift_apps_deploymentconfigs_last_failed_rollout_time` metric now has the correct `namespace` label set. (**BZ#2012770**)

Operator Lifecycle Manager (OLM)

- Before this update, default catalog sources for the `marketplace-operator` did not tolerate tainted nodes and the `CatalogSource` pod would only have the default settings for tolerations, `nodeSelector`, and `priorityClassName`. With this update, the `CatalogSource` specification now includes the optional `spec.grpcPodConfig` field that can override tolerations, `nodeSelector`, and `priorityClassName` for the pod. (**BZ#1927478**)
- Before this update, the `csv_suceeded` metric would be lost when the OLM Operator was restarted. With this update, the `csv_succeeded` metric is emitted at the beginning of the OLM Operator's startup logic. (**BZ#1927478**)
- Before this update, the `oc adm catalog mirror` command did not set minimum and maximum values for the `--max-icsp-size` flag. As a result, the field accepted values that were less than zero or were too large. With this update, values are limited to sizes greater than zero and less than 250001. Values outside of this range fail validation. (**BZ#1972962**)
- Before this update, bundled images did not contain the related images needed for Operator deployment in file-based catalogs. As a result, images were not mirrored to disconnected clusters unless specified in the `relatedImages` field of the ClusterServiceVersion (CSV). With this update, the `opm render` command adds the CSV Operator images to the `relatedImages` file when the file-based catalog bundle image is rendered. The images necessary for Operator deployment are now mirrored to disconnected clusters even if they are not listed in the `relatedImages` field of the CSV. (**BZ#2002075**)
- Before this update, it could take up to 15 minutes for Operators to perform `skipRange` updates. This was a known issue that could be resolved if cluster administrators deleted the `catalog-operator` pod in the `openshift-operator-lifecycle-manager` namespace. This caused

the pod to be automatically recreated and triggered the `skipRange` upgrade. With this update, obsolete API calls have been fixed in Operator Lifecycle Manager (OLM), and `skipRange` updates trigger immediately. (**BZ#2002276**)

- Occasionally, update events on clusters would happen at the same time that Operator Lifecycle Manager (OLM) modified an object from the lister cache. This caused concurrent map writes. This fix updates OLM so it no longer modifies objects retrieved from the lister cache. Instead, OLM modifies a copy of the object where applicable. As a result, OLM no longer experiences concurrent map writes. (**BZ#2003164**)
- Previously, Operator Lifecycle Manager (OLM) could not establish gRPC connections to catalog source pods that were only reachable through a proxy. If a catalog source pod was behind a proxy, OLM could not connect to the proxy and the hosted Operator content was unavailable for installation. This bug fix introduces a `GRPC_PROXY` environment variable that defines a proxy that OLM uses to establish connections to gRPC catalog sources. As a result, OLM can now be configured to use a proxy when connecting to gRPC catalog sources. (**BZ#2011927**)
- Previously, skipped bundles were not verified to be members of the same package. Bundles could skip across packages, which broke upgrade chains. This bug fix adds validation to ensure skipped bundles are in the same package. As a result, no bundle can skip bundles in another package, and upgrade graphs no longer break across packages. (**BZ#2017327**)
- In the `CatalogSource` object, the `RegistryServiceStatus` field stores service information that is used to generate an address that Operator Lifecycle Manager (OLM) relies on to establish a connection with the associated pod. If the `RegistryServiceStatus` field is not nil and is missing the namespace, name, and port information for its service, OLM is unable to recover until the associated pod has an invalid image or spec. With this bug fix, when reconciling a catalog source, OLM now ensures that the `RegistryServiceStatus` field of the `CatalogSource` object is valid and updates its status to reflect the change. Additionally, this address is stored within the status of the catalog source in the `status.GRPCConnectionState.Address` field. If the address changes, OLM updates this field to reflect the new address. As a result, the `.status.connectionState.address` field of a catalog source should no longer be nil. (**BZ#2026343**)

OpenShift API server

OpenShift Update Service

Red Hat Enterprise Linux CoreOS (RHCOS)

- Previously, when the RHCOS live ISO added a UEFI boot entry for itself, it assumed the existing UEFI boot entry IDs were consecutive, thereby causing the live ISO to fail in the UEFI firmware when booting on systems with non-consecutive boot entry IDS. With this fix, the RHCOS live ISO no longer adds a UEFI boot entry for itself and the ISO boots successfully. (**BZ#2006690**)
- To help you determine whether a user-provided image was already booted, information has been added on the terminal console describing when the machine was provisioned through Ignition and whether a user Ignition configuration was provided. This allows you to verify that Ignition ran when you expected it to. (**BZ#2016004**)

- Previously, when reusing an existing statically keyed LUKS volume during provisioning, the encryption key was not correctly written to disk and Ignition would fail with a "missing persisted keyfile" error. With this fix, Ignition now correctly writes keys for reused LUKS volumes so that existing statically keyed LUKS volumes can be reused during provisioning. (**BZ#2043296**)
- Previously, `ostree-finalize-staged.service` failed while upgrading a Red Hat Enterprise Linux CoreOS (RHCOS) node to 4.6.17. To fix this, the sysroot code now ignores any irregular or non-symlink files in `/etc`. (**BZ#1945274**)
- Previously, `initramfs` files were missing udev rules for by-id symlinks of attached SCSI devices. Because of this, Ignition configuration files that referenced these symlinks would result in a failed boot of the installed system. With this update, the `63-scsi-sg3_symlink.rules` for SCSI rules are added in dracut. (**BZ#1990506**)
- Previously, on bare-metal machines, a race condition occurred between `system-rfkill.service` and `ostree-remount.service`. Consequently, the `ostree-remount` service failed and the node operating system froze during the boot process. With this update, the `/sysroot/` directory is now read-only. As a result, the issue no longer occurs. (**BZ#1992618**)
- Previously, Red Hat Enterprise Linux CoreOS (RHCOS) live ISO boots added a UEFI boot entry, prompting a reboot on systems with a TPM. With this update, the RHCOS live ISO no longer adds a UEFI boot entry so the ISO does not initiate a reboot after first boot. (**BZ#2004449**)

Performance Addon Operator

- The spec.cpu.reserved` flag might not be correctly set by default if `spec.cpu.isolated` is the only parameter defined in `PerformanceProfile`. You must set the settings for both `spec.cpu.reserved` and `spec.cpu.isolated` in the `PerformanceProfile`. The sets must not overlap and the sum of all CPUs mentioned must cover all CPUs expected by the workers in the target pool. (**BZ#1986681**)
- Previously, the `oc adm must-gather` tool failed to collect node data if the `gather-sysinfo` binary was missing in the image. This was caused by a missing `COPY` statement in the Dockerfile. To avoid this issue, you must add the necessary `COPY` statements to the Dockerfile to generate and copy the binaries. (**BZ#2021036**)
- Previously, the Performance Addon Operator downloaded its image from the registry without checking whether it was available on the CRI-O cache. Consequently, the Performance Addon Operator failed to start if it could not reach the registry, or if the download timed out. With this update, the Operator only downloads its image from the registry if it cannot pull the image from the CRI-O cache. (**BZ#2021202**)
- When upgrading OpenShift Container Platform to version 4.10, any comment (`#comment`) in the tuned profile that does not start at the beginning of the line causes a parsing error. Performance Addon Operator issues can be solved by upgrading it to the same level (4.10) as OpenShift Container Platform. Comment-related errors can be worked around by putting all comments on a single line, with the # character at the start of the line. (**BZ#2059934**)

Routing

- Previously, if the cluster administrator provided a default ingress certificate that was missing

the newline character for the last line, the OpenShift Container Platform router would write out a corrupt PEM file for HAProxy. Now, it writes out a valid PEM file even if the input is missing a newline character. (**BZ#1894431**)

- Previously, a route created where the combined name and namespace for the DNS segment was greater than 63 characters long would be rejected. This expected behavior could cause problems integrating with upgraded versions of OpenShift Container Platform. Now, an annotation allows non-conformant DNS hostnames. With `AllowNonDNSCompliantHostAnnotation` set to `true`, the non-conformant DNS hostname, or one longer than 63 characters, is allowed. (**BZ#1964112**)
- Previously, the Cluster Ingress Operator would not create wildcard DNS records for Ingress Controllers when the cluster's `ControlPlaneTopology` was set to `External`. In Hypershift clusters where the `ControlPlaneTopology` was set to `External` and the Platform was AWS, the Cluster Ingress Operator never became available. This updates limits the disabling of DNS updates when the `ControlPlaneTopology` is `External` and the platform is IBM Cloud. As a result, wildcard DNS entries are created for Hypershift clusters running on AWS. (**BZ#2011972**)
- Previously, the cluster ingress router was blocked from working because the Ingress Operator failed to configure a wildcard DNS record for the cluster ingress router on Azure Stack Hub IPI. With this fix, the Ingress Operator now uses the configured ARM endpoint to configure DNS on Azure Stack Hub IPI. As a result, the cluster ingress router now works properly. (**BZ#2032566**)
- Previously, the cluster-wide prox configuration could not accept IPv6 addresses for the `noProxy` setting. Consequently, it was impossible to install a cluster whose configuration was having `noProxy` with IPv6 addresses. With this update, the Cluster Network Operator is now able to parse IPv6 addresses for the `noProxy` setting of the cluster-wide proxy resource. As a result, it is now possible to exclude IPv6 addresses for the `noProxy` setting. (**BZ#1939435**)
- Before OpenShift Container Platform 4.8, the IngressController API did not have any subfields under the `status.endpointPublishingStrategy.hostNetwork` and `status.endpointPublishingStrategy.nodePort` fields. These fields could be null even if the `spec.endpointPublishingStrategy.type` was set to `HostNetwork` or `NodePortService`. In OpenShift Container Platform 4.8, the `status.endpointPublishingStrategy.hostNetwork.protocol` and `status.endpointPublishingStrategy.nodePort.protocol` subfields were added, and the Ingress Operator set default values for these subfields when the Operator admitted or re-admitted an IngressController that specified the "HostNetwork" or "NodePortService" strategy type. With this bug, however, the Operator ignored updates to these spec fields, and updating `spec.endpointPublishingStrategy.hostNetwork.protocol` or `spec.endpointPublishingStrategy.nodePort.protocol` to `PROXY` to enable proxy protocol on an existing IngressController had no effect. To work around this issue, it was necessary to delete and recreate the IngressController to enable PROXY protocol. With this update, the Ingress Operator is changed so that it correctly updates the status fields when `status.endpointPublishingStrategy.hostNetwork` and `status.endpointPublishingStrategy.nodePort` are null and when the IngressController spec fields specify proxy protocol with the `HostNetwork` or `NodePortService` endpoint publishing strategy type. As a result, setting `spec.endpointPublishingStrategy.hostNetwork.protocol` or `spec.endpointPublishingStrategy.nodePort.protocol` to `PROXY` now takes proper effect on

upgraded clusters. (**BZ#1997226**)

Samples

- Before this update, if the Cluster Samples Operator encountered an `APIServerConflictError` error, it reported `sample-operator` as having `Degraded status` until it recovered. Momentary errors of this type were not unusual during upgrades but caused undue concern for administrators monitoring the Operator status. With this update, if the Operator encounters a momentary error, it no longer indicates `openshift-samples` as having `Degraded status` and tries again to connect to the API server. Momentary shifts to `Degraded status` no longer occur. (BZ#1993840)
- Before this update, various allowed and blocked registry configuration options in the cluster image configuration might prevent the Cluster Samples Operator from creating image streams. As a result, the samples operator might mark itself as degraded, which impacted the general OpenShift Container Platform install and upgrade status.

  In various circumstances, the management state of the Cluster Samples Operator can make the transition to `Removed`. With this update, these circumstances now include when the image controller configuration parameters prevent the creation of image streams by using either the default image registry or the image registry specified by the `samplesRegistry` setting. The Operator status now also indicates that the cluster image configuration is preventing the creation of the sample image streams. (BZ#2002368)

Storage

- Previously, the Local Storage Operator (LSO) took a long time to delete orphaned persistent volumes (PVs) due to the accumulation of a 10-second delay. With this update, the LSO does not use the 10-second delay, PVs are deleted promptly, and local disks are made available for new persistent volume claims sooner. (**BZ#2001605**)
- Previously, Manila error handling would degrade the Manila Operator, and the cluster. Errors are now treated as non-fatal so that the Manila Operator is disabled, rather than degrading the cluster. (**BZ#2001620**)
- In slower cloud environments, such as when using Cinder, the cluster might become degraded. Now, OpenShift Container Platform accommodates slower environments so that the cluster does not become degraded. (**BZ#2027685**)

Telco Edge

- If a generated policy has a complianceType of `mustonlyhave`, Operator Lifecycle Manager (OLM) updates to metadata are then reverted as the policy engine restores the 'expected' state of the CR. Consequently, OLM and the policy engine continuously overwrite the metadata of the CR under conflict. This results in high CPU usage. With this update, OLM and the policy engine no longer conflict, which reduces CPU usage. (**BZ#2009233**)
- Previously, user-supplied fields in the `PolicyGenTemplate` overlay were not copied to generated manifests if the field did not exist in the base source CR. As a result, some user content was lost. The `policyGen` tool is now updated to support all user supplied fields. (**BZ#2028881**)

- Previously, DNS lookup failures might cause the Cluster Baremetal Operator to continually fail when installed on unsupported platforms. With this update, the Operator remains disabled when installed on an unsupported platform. (**BZ#2025458**)

Web console (Administrator perspective)

Web console (Developer perspective)

- Before this update, resources in the **Developer** perspective of the web console had invalid links to details about that resource. This update resolves the issue. It creates valid links so that users can access resource details. (BZ#2000651)
- Before this update, you could only specify a subject in the `SinkBinding` form by label, not by name. With this update, you can use a drop-down list to select whether to specify a subject by name or label. (BZ#2002266)
- Before this update, the web terminal icon was available in the web console's banner head only if you installed the Web Terminal Operator in the `openShift-operators` namespace. With this update, the terminal icon is available regardless of the namespace where you install the Web Terminal Operator. (2006329)
- Before this update, the service binding connector did not appear in topology if you used a `resource` property rather than a `kind` property to define a `ServiceBinding` custom resource (CR). This update resolves the issue by reading the CR's `resource` property to display the connector on the topology. (BZ#2013545)
- Before this update, the name input fields used a complex and recursive regular expression to validate user inputs. This regular expression made name detection very slow and often caused errors. This update resolves the issue by optimizing the regular expression and avoiding recursive matching. Now, name detection is fast and does not cause errors. (BZ#2014497)
- Before this update, feature flag gating was missing from some extensions contributed by the knative plug-in. Although this issue did not affect what was displayed, these extensions ran unnecessarily, even if the serverless operator was not installed. This update resolves the issue by adding feature flag gating to the extensions where it was missing. Now, the extensions do not run unnecessarily. (BZ#2016438)
- Before this update, if you repeatedly clicked links to get details for resources such as custom resource definitions or pods and the application encountered multiple code reference errors, it failed and displayed a `t is not a function` error. This update resolves the issue. When an error occurs, the application resolves a code reference and stores the resolution state so that it can correctly handle additional errors. The application no longer fails when code reference errors occur. (BZ#2017130)
- Before this update, users with restricted access could not access their config map in a shared namespace to save their user settings on a cluster and load them in another browser or machine. As a result, user preferences such as pinned navigation items were only saved in the local browser storage and not shared between multiple browsers. This update resolves the issue: The web console Operator automatically creates RBAC rules so that each user can save these settings to a config map in a shared namespace and more easily switch between browsers. (BZ#2018234)

- Before this update, trying to create connections between virtual machines (VMs) in the **Topology** view failed with an "Error creating connection" message. This issue happened because this action relied on a method that did not support custom resource definition (CRDs). This update resolves the issue by adding support for CRDs. Now you can create connections between VMs. (BZ#2020904)
- Before this update, the tooltip for tasks in the **PipelineRun details** showed misleading information. It showed the time elapsed since the task ran, not how long they ran. For example, it showed 122 hours for a task that ran for a couple of seconds 5 days ago. With this update, the tooltip shows the duration of the task. (BZ#2011368)

Technology Preview features

Some features in this release are currently in Technology Preview. These experimental features are not intended for production use. Note the following scope of support on the Red Hat Customer Portal for these features:

Technology Preview Features Support Scope

In the table below, features are marked with the following statuses:

- **TP**: *Technology Preview*
- **GA**: *General Availability*
- **-**: *Not Available*
- **DEP**: *Deprecated*

Table 2. Technology Preview tracker

Feature	OCP 4.8	OCP 4.9	OCP 4.10
Precision Time Protocol (PTP) hardware configured as ordinary clock	TP	GA	GA
PTP single NIC hardware configured as boundary clock	-	-	TP
PTP events with ordinary clock	-	-	TP
`oc` CLI plug-ins	GA	GA	GA
CSI Volumes in OpenShift Builds	-	-	TP
Service Binding	TP	TP	GA
Raw Block with Cinder	GA	GA	GA
CSI volume expansion	TP	TP	TP
CSI AliCloud Disk Driver Operator	-	-	GA
CSI Azure Disk Driver Operator	TP	TP	GA
CSI Azure File Driver Operator	-	-	TP
CSI Azure Stack Hub Driver Operator	-	GA	GA

Feature	OCP 4.8	OCP 4.9	OCP 4.10
CSI GCP PD Driver Operator	GA	GA	GA
CSI IBM VPC Block Driver Operator	-	-	GA
CSI OpenStack Cinder Driver Operator	GA	GA	GA
CSI AWS EBS Driver Operator	TP	GA	GA
CSI AWS EFS Driver Operator	-	TP	GA
CSI automatic migration	TP	TP	TP
CSI inline ephemeral volumes	TP	TP	TP
CSI vSphere Driver Operator	TP	TP	GA
Shared Resource CSI Driver	-	-	TP
Automatic device discovery and provisioning with Local Storage Operator	TP	TP	TP
OpenShift Pipelines	GA	GA	GA
OpenShift GitOps	GA	GA	GA
OpenShift sandboxed containers	TP	TP	GA
Vertical Pod Autoscaler	GA	GA	GA
Cron jobs	GA	GA	GA
PodDisruptionBudget	GA	GA	GA
Adding kernel modules to nodes with kvc	TP	TP	TP
Egress router CNI plug-in	GA	GA	GA
Scheduler profiles	TP	GA	GA
Non-preempting priority classes	TP	TP	TP
Kubernetes NMState Operator	TP	TP	GA
Assisted Installer	TP	TP	TP
AWS Security Token Service (STS)	GA	GA	GA
Kdump	TP	TP	TP
OpenShift Serverless	GA	GA	GA
OpenShift on ARM platforms	-	-	GA
Serverless functions	TP	TP	TP
Data Plane Development Kit (DPDK) support	TP	GA	GA
Memory Manager feature	-	TP	TP
CNI VRF plug-in	TP	GA	GA
Cluster Cloud Controller Manager Operator	-	GA	GA
Cloud controller manager for Alibaba Cloud	-	-	TP
Cloud controller manager for Amazon Web Services	-	TP	TP

Feature	OCP 4.8	OCP 4.9	OCP 4.10
Cloud controller manager for Google Cloud Platform	-	-	TP
Cloud controller manager for IBM Cloud	-	-	TP
Cloud controller manager for Microsoft Azure	-	TP	TP
Cloud controller manager for Microsoft Azure Stack Hub	-	GA	GA
Cloud controller manager for Red Hat OpenStack Platform (RHOSP)	-	TP	TP
Cloud controller manager for VMware vSphere	-	-	TP
Driver Toolkit	TP	TP	TP
Special Resource Operator (SRO)	-	TP	TP
Simple Content Access	-	TP	GA
Node Health Check Operator	-	TP	TP
Network bound disk encryption (Requires Clevis, Tang)	-	GA	GA
MetalLB Operator	-	GA	GA
CPU manager	-	-	TP
Pod-level bonding for secondary networks	-	-	TP
IPv6 dual stack	-	GA	GA
Multicluster console	-	-	TP
Selectable Cluster Inventory	-	-	TP
Hyperthreading-aware CPU manager policy	-	-	TP
Dynamic Plug-ins	-	-	TP
Hybrid Helm Operator	-	-	TP
Alert routing for user-defined projects monitoring	-	-	TP
Disconnected mirroring with the oc-mirror CLI plug-in	-	-	TP
Mount shared entitlements in BuildConfigs in RHEL	-	-	TP
Mount shared secrets in RHEL	-	-	GA
Support for RHOSP DCN	-	-	TP
Support for external cloud providers for clusters on RHOSP	-	-	TP
OVS hardware offloading for clusters on RHOSP	-	-	TP

Known issues

- In OpenShift Container Platform 4.1, anonymous users could access discovery endpoints. Later releases revoked this access to reduce the possible attack surface for security exploits

because some discovery endpoints are forwarded to aggregated API servers. However, unauthenticated access is preserved in upgraded clusters so that existing use cases are not broken.

If you are a cluster administrator for a cluster that has been upgraded from OpenShift Container Platform 4.1 to 4.8, you can either revoke or continue to allow unauthenticated access. It is recommended to revoke unauthenticated access unless there is a specific need for it. If you do continue to allow unauthenticated access, be aware of the increased risks.

If you have applications that rely on unauthenticated access, they might receive HTTP `403` errors if you revoke unauthenticated access.

Use the following script to revoke unauthenticated access to discovery endpoints:

```
## Snippet to remove unauthenticated group from all the cluster role bindings
$ for clusterrolebinding in cluster-status-binding discovery system:basic-user
system:discovery system:openshift:discovery ;
do
### Find the index of unauthenticated group in list of subjects
index=$(oc get clusterrolebinding ${clusterrolebinding} -o json | jq
'select(.subjects!=null) | .subjects | map(.name=="system:unauthenticated") |
index(true)');
### Remove the element at index from subjects array
oc patch clusterrolebinding ${clusterrolebinding} --type=json --patch "[{'op':
'remove','path': '/subjects/$index'}]";
done
```

This script removes unauthenticated subjects from the following cluster role bindings:

- `cluster-status-binding`
- `discovery`
- `system:basic-user`
- `system:discovery`
- `system:openshift:discovery`

(**BZ#1821771**)

- The `oc annotate` command does not work for LDAP group names that contain an equal sign (`=`), because the command uses the equal sign as a delimiter between the annotation name and value. As a workaround, use `oc patch` or `oc edit` to add the annotation. (**BZ#1917280**)
- When upgrading to OpenShift Container Platform 4.10, the Cluster Version Operator blocks the upgrade for approximately five minutes while failing precondition checks. The error text, which says `It may not be safe to apply this update`, might be misleading. This error occurs when one or multiple precondition checks fail. In some situations, these precondition checks might only fail for a short period of time, for example, during an etcd backup. In these situations, the Cluster Version Operator and corresponding Operators will, by design,

automatically resolve the failing precondition checks and the CVO successfully starts the upgrade.

Users should check the status and conditions of their Cluster Operators. If the `It may not be safe to apply this update` error is displayed by the Cluster Version Operator, these statuses and conditions will provide more information about the severity of the message. For more information, see **BZ#1999777**, **BZ#2061444**, **BZ#2006611**.

- The assignment of egress IP addresses to control plane nodes with the egress IP feature is not supported on a cluster provisioned on Amazon Web Services (AWS). (**BZ#2039656**)
- Previously, there was a race condition between Red Hat OpenStack Platform (RHOSP) credentials secret creation and `kube-controller-manager` startup. As a result, Red Hat OpenStack Platform (RHOSP) cloud provider would not be configured with RHOSP credentials and would break support when creating Octavia load balancers for `LoadBalancer` services. To work around this, you must restart the `kube-controller-manager` pods by deleting the pods manually from the manifests. When you use the workaround, the `kube-controller-manager` pods restart and RHOSP credentials are properly configured. (**BZ#2004542**)
- The ability to delete operands from the web console using the `delete all operands` option is currently disabled. It will be re-enabled in a future version of OpenShift Container Platform. For more information, see BZ#2012120 and BZ#2012971.
- This release contains a known issue with Jenkins. If you customize the hostname and certificate of the OpenShift OAuth route, Jenkins no longer trusts the OAuth server endpoint. As a result, users cannot log in to the Jenkins console if they rely on the OpenShift OAuth integration to manage identity and access.

  Workaround: See the Red Hat Knowledge base solution, Deploy Jenkins on OpenShift with Custom OAuth Server URL. (**BZ#1991448**)

- This release contains a known issue with Jenkins. The `xmlstarlet` command line toolkit, which is required to validate or query XML files, is missing from this RHEL-based image. This issue impacts deployments that do not use OpenShift OAuth for authentication. Although OpenShift OAuth is enabled by default, users can disable it.

  Workaround: Use OpenShift OAuth for authentication. (BZ#2055653)

- Google Cloud Platform (GCP) UPI installation fails when the instance group name is longer than the maximum size of 64 characters. You are restricted in the naming process after adding the "-instance-group" suffix. Shorten the suffix to "-ig" to reduce the number of characters. (**BZ#1921627**)
- For clusters that run on RHOSP and use Kuryr, a bug in the OVN Provider driver for Octavia can cause load balancer listeners to be stuck in a `PENDING_UPDATE` state while the load balancer that they are attached to remains in an `ACTIVE` state. As a result, the `kuryr-controller` pod can crash. To resolve this problem, update RHOSP to a version that includes a fix as soon as possible. (**BZ#20004542**)
- If an incorrect network is specified in the vSphere `install-config.yaml` file, then an error message from Terraform is generated after a while. Add a check during the creation of

manifests to notify the user if the network is invalid. (**BZ#1956776**)

- The Special Resource Operator (SRO) might fail to install on Google Cloud Platform due to a software-defined network policy. As a result, the simple-kmod pod is not created. (**BZ#1996916**)
- Currently, idling a stateful set is unsupported when you run `oc idle` for a service that is mapped to a stateful set. There is no known workaround at this time. (**BZ#1976894**)
- The China (Nanjing) and UAE (Dubai) regions of Alibaba Cloud International Portal accounts do not support installer-provisioned infrastructure (IPI) installations. The China (Guangzhou) and China (Ulanqab) regions do not support a Server Load Balancer (SLB) if using Alibaba Cloud International Portal accounts and, therefore, also do not support IPI installations. (**BZ#2048062**)
- The Korea (Seoul) `ap-northeast-2` region of Alibaba Cloud does not support installer-provisioned infrastructure (IPI) installations. The Korea (Seoul) region does not support a Server Load Balancer (SLB) and, therefore, also does not support IPI installations. If you want to use OpenShift Container Platform in this region, contact Alibaba Cloud. (**BZ#2062525**)
- Currently, the **Knative Serving - Revision CPU, Memory, and Network usage** and **Knative Serving - Revision Queue proxy Metrics** dashboards are visible to all the namespaces, including those that do not have Knative services. (BZ#2056682)
- Currently, in the **Developer** perspective, the **Observe** dashboard opens for the most recently viewed workload rather than the one you selected in the **Topology** view. This issue happens because the session uses the Redux store rather than the query parameters in the URL. (BZ#2052953)
- Currently, the **ProjectHelmChartRepository** custom resource (CR) does not show up in the cluster because the API schema for this CR has not been initialized in the cluster yet. (BZ#2054197)
- Currently, while running high-volume pipeline logs, the auto-scroll functionality does not work and logs are stuck showing older messages. This issue happens because running high-volume pipeline logs generates a large number of calls to the `scrollIntoView` method. (BZ#2014161)
- Currently, when you use the **Import from Git** form to import a private Git repository, the correct import type and a builder image are not identified. This issue happens because the secret to fetch the private repository details is not decoded. (**BZ#2053501**)
- During an upgrade of the monitoring stack, Prometheus and Alertmanager might become briefly unavailable. No workaround for this issue is necessary because the components will be available after a short time has passed. No user intervention is required. (**BZ#203059**)
- For this release, monitoring stack components have been updated to use TLS authentication for metrics collection. However, sometimes Prometheus tries to keep HTTP connections to metrics targets open using expired TLS credentials even after new ones have been provided. Authentication errors then occur, and some metrics targets become unavailable. When this issue occurs, a `TargetDown` alert will fire. To work around this issue, restart the pods that are reported as down. (**BZ#2033575**)
- For this release, the number of Alertmanager replicas in the monitoring stack was reduced

from three to two. However, the persistent volume claim (PVC) for the removed third replica is not automatically removed as part of the upgrade process. After the upgrade, an administrator can remove this PVC manually from the Cluster Monitoring Operator. (**BZ#2040131**)

- Previously, the `oc adm must-gather` tool did not collect performance specific data when more than one `--image` argument was supplied. Files, including node and performance related files, were missing when the operation finished. The issue affects OpenShift Container Platform versions between 4.7 and 4.10. This issue can be resolved by executing the `oc adm must-gather` operation twice, once for each image. As a result, all expected files can be collected. (**BZ#2018159**)
- When using the Technology Preview oc-mirror CLI plug-in, there is a known issue that can occur when updating your cluster after mirroring an updated image set to the mirror registry. If a new version of an Operator is published to a channel by deleting the previous version of that Operator and then replacing it with a new version, an error can occur when applying the generated `CatalogSource` file from the oc-mirror plug-in, because the catalog is seen as invalid. As a workaround, delete the previous catalog image from the mirror registry, generate and publish a new differential image set, and then apply the `CatalogSource` file to the cluster. You must follow this workaround each time you publish a new differential image set, until this issue is resolved. (**BZ#2060837**)
- The processing of the `StoragePVC` custom resource during the GitOps ZTP flow does not exclude the `volume.beta.kubernetes.io/storage-class` annotation when a user does not include a value for it. This annotation causes the `spec.storageClassName` field to be ignored. To avoid this, set the desired `StorageClass` name in the `volume.beta.kubernetes.io/storage-class` annotation within your `PolicyGenTemplate` when using a `StoragePVC` custom resource. (**BZ#2060554**)
- Removing a Bidirectional Forwarding Detection (BFD) custom profile enabled on a border gateway protocol (BGP) peer resource does not disable the BFD. Instead, the BGP peer starts using the default BFD profile. To disable BFD from a BGP peer resource, delete the BGP peer configuration and recreate it without a BFD profile. (**BZ#2050824**)
- For clusters that run on RHOSP and use Mellanox NICs as part of a single-root I/O virtualization configuration (SR-IOV), you may not be able to create a pod after you start one, restart the SR-IOV device plugin, and then stop the pod. No workaround is available for this issue.
- OpenShift Container Platform supports deploying an installer-provisioned cluster without a DHCP server. However, without a DHCP server, the bootstrap VM does not receive an external IP address for the `baremetal` network. To assign an IP address to the bootstrap VM, see Assigning a bootstrap VM an IP address on the baremetal network without a DHCP server. (**BZ#2048600**)
- OpenShift Container Platform supports deploying an installer-provisioned cluster with static IP addresses on the `baremetal` network for environments without a DHCP server. If a DHCP server is present, nodes might retrieve an IP address from the DHCP server on reboot. To prevent DHCP from assigning an IP address to a node on reboot, see Preventing DHCP from assigning an IP address on node reboot. (**BZ#2036677**)
- The RHCOS kernel experiences a soft lockup and eventually panics due to a bug in the Netfilter module. A fix is planned to resolve this issue in a future z-stream release of

OpenShift Container Platform. (**BZ#2061445**)

- The Kubelet service monitor scrape interval is currently set to a hard-coded value. This means that there is less available CPU resources for workloads. (**BZ#2035046**)
- Deploying a single node OpenShift cluster for vRAN Distributed Units can take up to 4 hours. (**BZ#2035036**)
- Currently, if an RHACM policy was enforced to a target cluster, you can create a `ClusterGroupUpgrade` CR including that enforced policy again as a `managedPolicy` to the same target cluster. This should not be possible. (**BZ#2044304**)
- If a `ClusterGroupUpgrade` CR has a `blockingCR` specified, and that `blockingCR` fails silently, for example if there is a typo in the list of clusters, the `ClusterGroupUpgrade` CR is applied even though the `blockingCR` is not applied in the cluster. (**BZ#2042601**)
- If a `ClusterGroupUpgrade` CR validation fails for any reason, for example, because of an invalid spoke name, no status is available for the `ClusterGroupUpgrade` CR, as if the CR is disabled. (**BZ#2040828**)
- Currently, for `ClusterGroupUpgrade` CR state changes, there is only a single condition type available - `Ready`. The `Ready` condition can have a status of `True` or `False` only. This does not reflect the range of states the `ClusterGroupUpgrade` CR can be in. (**BZ#2042596**)
- When deploying a multi-node cluster on bare-metal nodes, the machine config pool (MCP) adds an additional CRI-O drop-in that circumvents the container mount namespace drop-in. This results in CRI-O being in the base namespace while the kubelet is in the hidden namespace. All containers fail to get any kubelet-mounted filesystems, such as secrets and tokens. (**BZ#2028590**)
- Installing RHACM 2.5.0 in the customized namespace causes the `infrastructure-operator` pod to fail due to insufficient privileges. (**BZ#2046554**)
- OpenShift Container Platform limits object names to 63 characters. If a policy name defined in a `PolicyGenTemplate` CR approaches this limit, the Cluster Group Upgrades Operator cannot create child policies. When this occurs, the parent policy will remain in a `NonCompliant` state. (**BZ#2057209**)
- In the default ZTP Argo CD configuration, cluster names cannot begin with `ztp`. Using names starting with `ztp` for clusters deployed with Zero Touch Provisioning (ZTP) results in provisioning not completing. As a workaround, ensure that either cluster names do not start with `ztp`, or adjust the Argo CD policy application namespace to a pattern that excludes the names of your clusters but still matches your policy namespace. For example, if your cluster names start with `ztp`, change the pattern in the Argo CD policy app configuration to something different, like `ztp-`. (**BZ#2049154**)
- During a spoke cluster upgrade, one or more reconcile errors is recorded in the container log. The number of errors corresponds to the number of child policies. The error causes no noticeable impact to the cluster. The following is an example of the reconcile error:

```
2022-01-21T00:14:44.697Z        INFO    controllers.ClusterGroupUpgrade Upgrade
is completed
2022-01-21T00:14:44.892Z        ERROR   controller-
runtime.manager.controller.clustergroupupgrade       Reconciler error
{"reconciler group": "ran.openshift.io", "reconciler kind":
"ClusterGroupUpgrade", "name": "timeout", "namespace": "default", "error":
"Operation cannot be fulfilled on clustergroupupgrades.ran.openshift.io
\"timeout\": the object has been modified; please apply your changes to the
latest version and try again"}
sigs.k8s.io/controller-
runtime/pkg/internal/controller.(*Controller).processNextWorkItem
        /go/pkg/mod/sigs.k8s.io/controller-
runtime@v0.9.2/pkg/internal/controller/controller.go:253
sigs.k8s.io/controller-
runtime/pkg/internal/controller.(*Controller).Start.func2.2
        /go/pkg/mod/sigs.k8s.io/controller-
runtime@v0.9.2/pkg/internal/controller/controller.go:214
```

(**BZ#2043301**)

- During a spoke cluster upgrade from 4.9 to 4.10, with heavy workload running, the `kube-apiserver` pod can take longer than the expected time to start. As a result, the upgrade does not complete and the `kube-apiserver` rolls back to the previous version. (**BZ#2064024**)
- If you deploy the AMQ Interconnect Operator, pods run on IPv4 nodes only. The AMQ Interconnect Operator is not supported on IPv6 nodes. (**ENTMQIC-3297**)

Asynchronous errata updates

Security, bug fix, and enhancement updates for OpenShift Container Platform 4.10 are released as asynchronous errata through the Red Hat Network. All OpenShift Container Platform 4.10 errata are available on the Red Hat Customer Portal. See the OpenShift Container Platform Life Cycle for more information about asynchronous errata.

Red Hat Customer Portal users can enable errata notifications in the account settings for Red Hat Subscription Management (RHSM). When errata notifications are enabled, users are notified through email whenever new errata relevant to their registered systems are released.

Red Hat Customer Portal user accounts must have systems registered and consuming OpenShift Container Platform entitlements for OpenShift Container Platform errata notification emails to generate.

This section will continue to be updated over time to provide notes on enhancements and bug fixes for future asynchronous errata releases of OpenShift Container Platform 4.10. Versioned asynchronous releases, for example with the form OpenShift Container Platform 4.10.z, will be detailed in subsections. In addition, releases in which the errata text cannot fit in the space provided by the advisory will be detailed in subsections that follow.

For any OpenShift Container Platform release, always review the instructions on updating your cluster properly.

RHSA-2022:0056 - OpenShift Container Platform 4.10.3 image release, bug fix, and security update advisory

Issued: 2022-03-10

OpenShift Container Platform release 4.10.3, which includes security updates, is now available. The list of bug fixes that are included in the update is documented in the RHSA-2022:0056 advisory. A secondary set of bug fixes can be found in the RHEA-2022:0748 advisory. The RPM packages that are included in the update are provided by the RHSA-2022:0055 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.3 container image list

RHBA-2022:0811 - OpenShift Container Platform 4.10.4 bug fix and security update

Issued: 2022-03-15

OpenShift Container Platform release 4.10.4, which includes security updates, is now available. The bug fixes that are included in the update are listed in the RHBA-2022:0811 advisory. The RPM packages that are included in the update are provided by the RHSA-2022:0810 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.4 container image list

Updating

To update an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster within a minor version by using the CLI for instructions.

RHBA-2022:0928 - OpenShift Container Platform 4.10.5 bug fix and security update

Issued: 2022-03-21

OpenShift Container Platform release 4.10.5, which includes security updates, is now available. The bug fixes that are included in the update are listed in the RHBA-2022:0928 advisory. The RPM packages that are included in the update are provided by the RHSA-2022:0927 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.5 container image list

Known issues

- There is an issue adding a cluster through zero touch provisioning (ZTP) with the name `ztp*`. Adding `ztp` as the name of a cluster causes a situation where `ArgoCD` deletes policies that ACM copies in to the cluster namespace. Naming the cluster with `ztp` leads to a reconciliation loop, and the policies will not be compliant. As a workaround, do not name clusters with `ztp` at the beginning of the name. By renaming the cluster, collision will stop the reconciliation loop and the policies will be compliant. (**BZ#2049154**)

Bug fixes

- Previously, the **Observer** dashboard from the **Topology** view in the **Developer** console opened to the last viewed workload rather than the selected one. With this update, the **Observe** dashboard in the **Developer** console always opens to the selected workload from the **Topology** view. (**BZ#2059805**)

Updating

To update an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster within a minor version by using the CLI for instructions.

RHBA-2022:1026 - OpenShift Container Platform 4.10.6 bug fix and security update

Issued: 2022-03-28

OpenShift Container Platform release 4.10.6, which includes security updates, is now available. The bug fixes that are included in the update are listed in the RHBA-2022:1026 advisory. The RPM packages that are included in the update are provided by the RHSA-2022:1025 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.6 container image list

Features

Updates from Kubernetes 1.23.5

This update contains changes from Kubernetes 1.23.3 up to 1.23.5. More information can be found in the following changelogs: 1.23.4 and 1.23.5

Bug fixes

- Previously, the query for subnets in Cisco ACI's neutron implementation, which is avaiable in Red Hat OpenStack Platform (RHOSP)-16, returned unexpected results for a given network. Consequently, the RHOSP `cluster-api-provider` could potentially try to provision instances with duplicated ports on the same subnet, which caused failed provision. With this update, an additional filter is added in the RHOSP `cluster-api-provider` to ensure there is only one port per subnet. As a result, it is now possible to deploy OpenShift Container Platform on

RHOSP-16 with Cisco ACI. (**BZ#2050064**)

- Previously, `oc adm must gather` fell back to the `oc adm inspect` command when the specified image could not run. Consequently, it was difficult to understand information from the logs when the fall back happened. With this update, the logs are improved to make it explicit when a fall back inspection is performed. As a result, the output of `oc adm must gather` is easier to understand. (**BZ#2049427**)
- Previously, the `oc debug node` command did not have timeout specified on idle. Consequently, the users were never logged out of the cluster. With this update, a `TMOUT` environment variable for debug pod has been added to counter inactivity timeout. As a result, the session will be automatically terminated after `TMOUT` inactivity. (**BZ#2060888**)
- Previously, the Ingress Operator performed health checks against the Ingress canary route. When the health check completed, the Ingress Operator did not close the TCP connection to the `LoadBalancer` because `keepalive` packets were enabled on the connection. While performing the next health check, a new connection was established to the `LoadBalancer` instead of using the existing connection. Consequently, this caused connections to accumulate on the `LoadBalancer`. Over time, this exhausted the number of connections on the `LoadBalancer`. With this update, Keepalive is disabled when connecting to the Ingress canary route. As a result, a new connection is made and closed each time the canary probe is run. While Keepalive is disabled, there is no longer an accumulation of established connections. (**BZ#2063283**)
- Previously, the sink for event sources in the `Trigger/Subscription` modal in the **Topology** UI showed all resources, irrespective of whether they were created as a standalone or an underlying resource included with back KSVC, Broker, or KameletBinding. Consequently, users could sink to the underlying addressable resources as they showed up in the sink drop-down menu. With this update, a resource filter has been added to show only standalone resource sink events. (**BZ#2059807**)

Updating

To update an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster within a minor version by using the CLI for instructions.

RHSA-2022:1162 - OpenShift Container Platform 4.10.8 bug fix and security update

Issued: 2022-04-07

OpenShift Container Platform release 4.10.8, which includes security updates, is now available. The bug fixes that are included in the update are listed in the RHSA-2022:1162 advisory. The RPM packages that are included in the update are provided by the RHBA-2022:1161 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.8 container image list

Removed features

Starting with OpenShift Container Platform 4.10.8, support for Google Cloud Platform Workload Identity has been removed from OpenShift Container Platform 4.10 for the image registry. This change is due to the discovery of an adverse impact to the image registry. For more information, see Image registry support for Google Cloud Platform Workload Identity removed.

Known issues

- Currently, the web console does not display virtual machine templates that are deployed to a custom namespace. Only templates deployed to the default namespace are displayed in the web console. As a workaround, avoid deploying templates to a custom namespace. (**BZ#2054650**)

Bug fixes

- Previously, the Infrastructure Operator could not provision X11- and X12-based systems due to validation errors created by the bare metal controller (BMC) when special characters such as question marks or equal signs were used in the `filename` parameters of URLs. With this update, the `filename` parameter is removed from the URL if the virtual media image is backed by a local file. (**BZ#2011626**)
- Previously, when cloning a virtual machine from a template, Operator-made changes reverted after dismissing the dialog box if the boot disk was edited and the storage class was changed. With this update, changes made to storage class remain set after closing the dialogue box. (**BZ#2049762**)
- Previously, the `startupProbe` field was added to a container's definition. As a result, `startupProbe` causes problems when creating a debug pod. With this update, `startupProbe` is removed by default from the debug pod by the `Expose --keep-startup flag` parameter, which is now set to false by default. (**BZ#2068474**)
- Previously, the Local Storage Operator (LSO) added an `OwnerReference` object to the persistent volumes (PV) it created, which sometimes caused an issue where a delete request for a PV could leave the PV in the `terminating` state while still attached to the pod. With this update, the LSO no longer creates an `OwnerReference` object and cluster administrators are now able to manually delete any unused PVs after a node is removed from the cluster. (**BZ#2065714**)

Updating

To update an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster within a minor version by using the CLI for instructions.

RHBA-2022:1241 - OpenShift Container Platform 4.10.9 bug fix update

Issued: 2022-04-12

OpenShift Container Platform release 4.10.9 is now available. The bug fixes that are included in the update are listed in the RHBA-2022:1241 advisory. The RPM packages that are included in the update are provided by the RHBA-2022:1240 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.9 container image list

Known issues

- When updating to OpenShift Container Platform 4.10.9, the etcd pod fails to start and the etcd Operator falls into a `degraded` state. A future version of OpenShift Container Platform will resolve this issue. For more information, see etcd pod is failing to start after updating OpenShift Container Platform 4.9.28 or 4.10.9 and Potential etcd data inconsistency issue in OCP 4.9 and 4.10.

Updating

To update an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster within a minor version by using the CLI for instructions.

RHSA-2022:1357 - OpenShift Container Platform 4.10.10 bug fix and security update

Issued: 2022-04-20

OpenShift Container Platform release 4.10.10 is now available. The bug fixes that are included in the update are listed in the RHSA-2022:1357 advisory. The RPM packages that are included in the update are provided by the RHBA-2022:1355 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.10 container image list

Bug fixes

- Previously, the cluster storage Operator credentials request for Amazon Web Services (AWS) did not include KMS statements. Consequently, persistent volumes (PVs) failed to deploy due to the inability to provide a key. With this update, the default credentials request for AWS now allows the mounting of encrypted volumes using customer-managed keys from KMS. Administrators who create credentials requests in manual mode with Cloud Credential Operator (CCO) must apply those changes manually. Other administrators should not be impacted by this change. (**BZ#2072191**)

Updating

To update an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster within a minor version by using the CLI for instructions.

RHBA-2022:1431 - OpenShift Container Platform 4.10.11 bug fix update

Issued: 2022-04-25

OpenShift Container Platform release 4.10.11 is now available. The bug fixes that are included in the update are listed in the RHBA-2022:1431 advisory. There are no RPM packages for this release.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.11 container image list

Bug fixes

- Previously, when cloning a virtual machine from a template, Operator-made changes reverted after dismissing the dialog box if the boot disk was edited and the storage class was changed. With this update, changes made to storage class remain set after closing the dialogue box. (**BZ#2049762**)

Updating

To upgrade an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster using the CLI for instructions.

RHBA-2022:1601 - OpenShift Container Platform 4.10.12 bug fix and security update

Issued: 2022-05-02

OpenShift Container Platform release 4.10.12, which includes security updates, is now available. The bug fixes that are included in the update are listed in the RHBA-2022:1601 advisory. The RPM packages that are included in the update are provided by the RHSA-2022:1600 advisory.

Space precluded documenting all of the container images for this release in the advisory. See the following article for notes on the container images in this release:

OpenShift Container Platform 4.10.12 container image list

Bug fixes

- Previously, the Infrastructure Operator could not provision X11- and X12-based systems. This was due to validation errors created by the bare metal controller (BMC) when special characters such as question marks or equal signs were used in the `filename` parameters of URLs. With this update, the `filename` parameter is removed from the URL if the virtual media image is backed by a local file. (**BZ#2011626**)

Updating

To upgrade an existing OpenShift Container Platform 4.10 cluster to this latest release, see Updating a cluster using the CLI for instructions.

Monitoring Redfish hardware events

Redfish hardware events is a Developer Preview feature only. Developer Preview features are not supported with Red Hat production service level agreements (SLAs) and are not functionally complete or production-ready. Do not use Developer Preview features for production or business-critical workloads. Developer Preview features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. These features might not have any documentation, and testing is limited. Red Hat might provide ways to submit feedback on Developer Preview features without an associated SLA.

About Redfish hardware events

Use Redfish hardware events to subscribe applications that run in your OpenShift Container Platform cluster to events that are generated on the underlying bare-metal host. The Redfish service publishes events on a node and transmits them on an advanced message queue to subscribed applications.

Redfish hardware events are based on the open Redfish standard that is developed under the guidance of the Distributed Management Task Force (DMTF). Redfish provides a secure industry-standard protocol with a REST API. The protocol is used for the management of distributed, converged or software-defined resources and infrastructure.

Hardware-related events published through Redfish includes:

- Breaches of temperature limits
- Server status
- Fan status

Begin using Redfish hardware events by deploying the Operator and subscribing your application to the service. The Operator installs and manages the lifecycle of the Redfish hardware event service.

Redfish hardware events work only with Redfish-capable devices on single-node clusters provisioned on bare-metal infrastructure.

How Redfish hardware events work

The Baremetal Hardware Event Proxy enables applications running on bare-metal clusters to respond quickly to Redfish hardware changes and failures such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. These hardware events are delivered over a reliable low-latency transport channel based on Advanced Message Queuing Protocol (AMQP). The latency of the messaging service is between 10 to 20 milliseconds.

The hardware event proxy provides a publish-subscribe service for the hardware events, where multiple applications can use REST APIs to subscribe and consume the events. The hardware event proxy supports hardware that complies with Redfish OpenAPI v1.8 or higher.

Redfish hardware events data flow

The following figure illustrates an example Redfish hardware events data flow. vDU is used as an example of an application interacting with Redfish events:

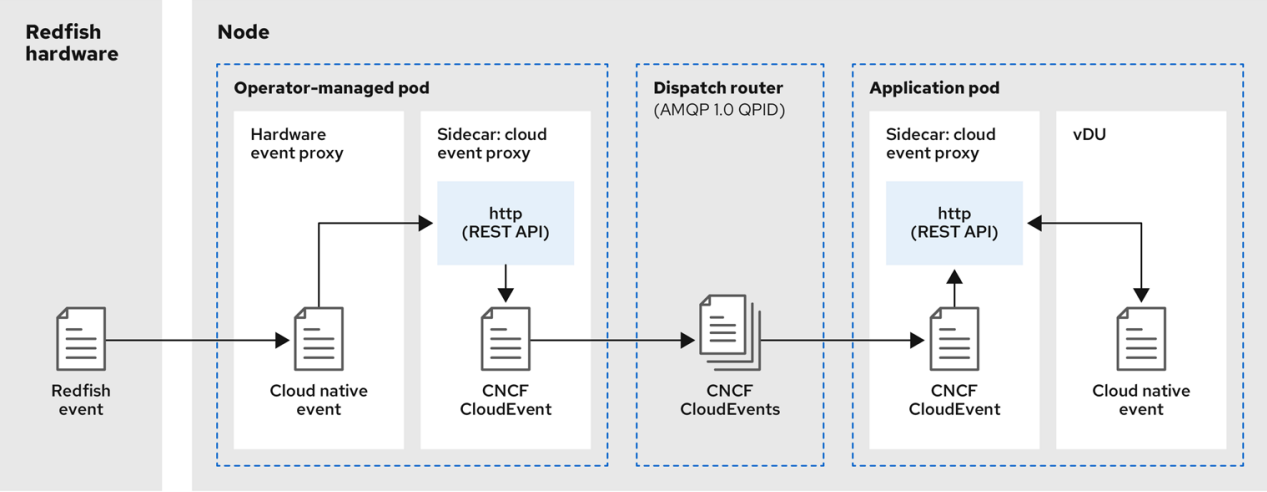


Figure 1. Redfish hardware events data flow

Operator-managed pod

The Operator uses custom resources to manage the pod containing the hardware event proxy and its components using the `HardwareEvent` CR.

Hardware event proxy

At startup, the hardware event proxy queries the Redfish API and downloads all the message registries, including custom registries. The hardware event proxy then begins to receive subscribed events from the Redfish hardware.

The proxy enables applications running on bare-metal clusters to respond quickly to Redfish hardware changes and failures such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. The events are reported using the `HardwareEvent` CR.

Cloud native event

Cloud native events (CNE) is a REST API specification for defining the format of event data.

CNCF CloudEvents

CloudEvents is a vendor-neutral specification developed by the Cloud Native Computing Foundation (CNCF) for defining the format of event data.

AMQP dispatch router

The dispatch router is responsible for the message delivery service between publisher and subscriber. AMQP 1.0 qpid is an open standard that supports reliable, high-performance, fully-symmetrical messaging over the internet.

Cloud event proxy sidecar

The cloud event proxy sidecar container image is based on the ORAN API specification and provides a publish-subscribe event framework for hardware events.

Redfish message parsing service

In addition to handling Redfish events, the Redfish hardware event proxy provides message parsing for events without a `Message` property. The proxy downloads all the Redfish message registries including vendor specific registries from the hardware when it starts. If an event does not contain a `Message` property, the proxy uses the Redfish message registries to construct the `Message` and `Resolution` properties and add them to the event before passing the event to the cloud events framework. This service allows Redfish events to have smaller message size and lower transmission latency.

Installing the upstream Hardware Event Proxy Operator

The Hardware Event Proxy Operator is not available in the `redhat-operators` Operators source for OpenShift Container Platform version 4.10. To install the Operator, see Installing upstream RAN operators.

Hardware Event Proxy Operator is not supported with Red Hat production service level agreements (SLAs) and is not functionally complete or production-ready. The Operator is provided for preview purposes only. Do not use Hardware Event Proxy Operator for production or business-critical workloads.

Installing the AMQ messaging bus

To pass Redfish hardware event notifications between publisher and subscriber on a node, you must install and configure an AMQ messaging bus to run locally on the node. You do this by installing the AMQ Interconnect Operator for use in the cluster.

Prerequisites

- Install the OpenShift Container Platform CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.

Procedure

- Install the AMQ Interconnect Operator to its own `amq-interconnect` namespace. See Installing the AMQ Interconnect Operator.

Verification

1. Verify that the AMQ Interconnect Operator is available and the required pods are running:

```
$ oc get pods -n amq-interconnect
```

Example output

```
NAME                                    READY   STATUS    RESTARTS   AGE
amq-interconnect-645db76c76-k8ghs       1/1     Running   0          23h
interconnect-operator-5cb5fc7cc-4v7qm   1/1     Running   0          23h
```

2. Verify that the required `hw-event-proxy` hardware event producer pod is running in the `openshift-hw-events` namespace:

```
$ oc get pods -n openshift-hw-events
```

Example output

```
NAME
READY   STATUS    RESTARTS   AGE
baremetal-hardware-event-proxy-operator-controller-manager-b889bb8d5-qwhvs
2/2     Running   0          25s
```

Subscribing to Redfish BMC hardware events for a cluster node

As a cluster administrator, you can subscribe to Redfish BMC events generated on a node in your cluster by creating a `BMCEventSubscription` custom resource (CR) for the node, creating a `HardwareEvent` CR for the event, and a `Secret` CR for the BMC.

Subscribing to Redfish hardware events

You can configure the baseboard management controller (BMC) to send Redfish hardware events to subscribed applications running in an OpenShift Container Platform cluster. Example Redfish events include an increase in device temperature, or removal of a device. Applications subscribe to Redfish hardware events using a REST API.

You can only create a `BMCEventSubscription` custom resource (CR) for physical hardware that supports Redfish and has a vendor interface set to `redfish` or `idrac-redfish`.

Use the `BMCEventSubscription` CR to subscribe to predefined Redfish events. The Redfish standard does not provide an option to create specific alerts and thresholds. For example, to receive an alert event when an enclosure's temperature exceeds 40° Celsius, you must manually configure the event according to the vendor's recommendations.

Perform the following procedure to subscribe to Redfish hardware events for the node using a `BMCEventSubscription` CR.

Prerequisites

- Install the OpenShift CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.
- Get the user name and password for the BMC.
- Deploy a bare-metal node with a RedFish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish events on the BMC.

Enabling Redfish events on specific hardware is outside the scope of this information. For more information about enabling Redfish events for your specific hardware, consult the BMC manufacturer documentation.

Procedure

1. Confirm that the node hardware has the Redfish `EventService` enabled by running the following `curl` command:

```
curl https://<bmc_ip_address>/redfish/v1/EventService --insecure -H 'Content-
Type: application/json' -u "<bmc_username>:<password>"
```

   where:

   bmc_ip_address

   is the IP address of the BMC where the Redfish events are generated.

Example output

```
{
   "@odata.context": "/redfish/v1/$metadata#EventService.EventService",
   "@odata.id": "/redfish/v1/EventService",
   "@odata.type": "#EventService.v1_0_2.EventService",
   "Actions": {
      "#EventService.SubmitTestEvent": {
         "EventType@Redfish.AllowableValues": ["StatusChange",
"ResourceUpdated", "ResourceAdded", "ResourceRemoved", "Alert"],
         "target":
"/redfish/v1/EventService/Actions/EventService.SubmitTestEvent"
      }
   },
   "DeliveryRetryAttempts": 3,
   "DeliveryRetryIntervalSeconds": 30,
   "Description": "Event Service represents the properties for the service",
   "EventTypesForSubscription": ["StatusChange", "ResourceUpdated",
"ResourceAdded", "ResourceRemoved", "Alert"],
   "EventTypesForSubscription@odata.count": 5,
   "Id": "EventService",
   "Name": "Event Service",
   "ServiceEnabled": true,
   "Status": {
      "Health": "OK",
      "HealthRollup": "OK",
      "State": "Enabled"
   },
   "Subscriptions": {
      "@odata.id": "/redfish/v1/EventService/Subscriptions"
   }
}
```

2. Get the hardware event proxy service route for the cluster by running the following command:

```
$ oc get route -n openshift-hw-events
```

Example output

```
NAME                    HOST/PORT
PATH   SERVICES                  PORT    TERMINATION           WILDCARD
hw-event-proxy          hw-event-proxy-cloud-native-events.apps.compute-
1.example.com                          hw-event-proxy-service    9087
edge                 None
```

3. Create a `BMCEventSubscription` resource to subscribe to the Redfish events:

   a. Save the following YAML in the `bmc_sub.yaml` file:

```
apiVersion: metal3.io/v1alpha1
kind: BMCEventSubscription
metadata:
  name: sub-01
  namespace: openshift-machine-api
spec:
   hostName: <hostname> ①
   destination: <proxy_service_url> ②
   context: ''
```

① Specifies the name or UUID of the worker node where the Redfish events are generated.

② Specifies the bare-metal hardware event proxy service, for example, https://hw-event-proxy-cloud-native-events.apps.compute-1.example.com/webhook.

b. Create the BMCEventSubscription CR:

```
$ oc create -f bmc_sub.yaml
```

4. Optional: To delete the BMC event subscription, run the following command:

```
$ oc delete -f bmc_sub.yaml
```

5. Optional: To manually create a Redfish event subscription without creating a BMCEventSubscription CR, run the following curl command, specifying the BMC username and password.

```
$ curl -i -k -X POST -H "Content-Type: application/json"  -d '{"Destination":
"https://<proxy_service_url>", "Protocol" : "Redfish", "EventTypes": ["Alert"],
"Context": "root"}' -u <bmc_username>:<password>
'https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions' v
```

where:

proxy_service_url

is the bare-metal hardware event proxy service, for example, https://hw-event-proxy-cloud-native-events.apps.compute-1.example.com/webhook.

bmc_ip_address

is the IP address of the BMC where the Redfish events are generated.

Example output

```
HTTP/1.1 201 Created
Server: AMI MegaRAC Redfish Service
Location: /redfish/v1/EventService/Subscriptions/1
Allow: GET, POST
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: X-Auth-Token
Access-Control-Allow-Headers: X-Auth-Token
Access-Control-Allow-Credentials: true
Cache-Control: no-cache, must-revalidate
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>; rel
=describedby
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>
Link: </redfish/v1/EventService/Subscriptions>; path=
ETag: "1651135676"
Content-Type: application/json; charset=UTF-8
OData-Version: 4.0
Content-Length: 614
Date: Thu, 28 Apr 2022 08:47:57 GMT
```

Querying Redfish hardware event subscriptions with curl

Some hardware vendors limit the amount of Redfish hardware event subscriptions. You can query the number of Redfish event subscriptions by using `curl`.

Prerequisites

- Get the user name and password for the BMC.
- Deploy a bare-metal node with a RedFish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish events on the BMC.

Procedure

1. Check the current subscriptions for the BMC by running the following `curl` command:

```
$ curl --globoff -H "Content-Type: application/json" -k -X GET --user
<bmc_username>:<password>
https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions
```

where:

bmc_ip_address
: is the IP address of the BMC where the Redfish events are generated.

Example output

```
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 435 100 435 0 0 399 0 0:00:01 0:00:01 --:--:-- 399
{
  "@odata.context":
"/redfish/v1/$metadata#EventDestinationCollection.EventDestinationCollection",
  "@odata.etag": ""
  1651137375 "",
  "@odata.id": "/redfish/v1/EventService/Subscriptions",
  "@odata.type": "#EventDestinationCollection.EventDestinationCollection",
  "Description": "Collection for Event Subscriptions",
  "Members": [
  {
    "@odata.id": "/redfish/v1/EventService/Subscriptions/1"
  }],
  "Members@odata.count": 1,
  "Name": "Event Subscriptions Collection"
}
```

In this example, a single subscription is configured: `/redfish/v1/EventService/Subscriptions/1`.

2. Optional: To remove the `/redfish/v1/EventService/Subscriptions/1` subscription with `curl`, run the following command, specifying the BMC username and password:

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\n" -k -u
<bmc_username>:<password >-H "Content-Type: application/json" -d '{}' -X DELETE
https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions/1
```

where:

bmc_ip_address
: is the IP address of the BMC where the Redfish events are generated.

Creating the Redfish HardwareEvent and Secret CRs

To start using bare-metal hardware events, create the a `HardwareEvent` custom resource (CR) for the host where the Redfish hardware is present. Hardware events and faults are reported in the `hw-event-proxy` logs.

Prerequisites

- Install the OpenShift CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.
- Install the Hardware Event Proxy Operator.
- Create a `BMCEventSubscription` CR for the BMC Redfish hardware.

Multiple `HardwareEvent` resources are not permitted.

Procedure

1. Create the `HardwareEvent` custom resource (CR):

   a. Save the following YAML in the `hw-event.yaml` file:

```
apiVersion: "event.redhat-cne.org/v1alpha1"
kind: "HardwareEvent"
metadata:
  name: "hardware-event"
spec:
  nodeSelector:
    node-role.kubernetes.io/hw-event: "" ①
  transportHost: "amqp://amq-router-service-name.amq-
namespace.svc.cluster.local" ②
  logLevel: "debug" ③
  msgParserTimeout: "10" ④
```

   ① Required. Use the `nodeSelector` field to target nodes with the specified label, for example, `node-role.kubernetes.io/hw-event: ""`.

   ② Required. AMQP host that delivers the events at the transport layer using the AMQP protocol.

   ③ Optional. The default value is `debug`. Sets the log level in hw-event-proxy logs. The following log levels are available: `fatal`, `error`, `warning`, `info`, `debug`, `trace`.

   ④ Optional. Sets the timeout value in milliseconds for the Message Parser. If a message parsing request is not responded to within the timeout duration, the original hardware event message is passed to the cloud native event framework. The default value is 10.

   b. Create the `HardwareEvent` CR:

```
$ oc create -f hardware-event.yaml
```

2. Create a BMC username and password `Secret` CR that enables the hardware events proxy to access the Redfish message registry for the bare-metal host.

   a. Save the following YAML in the `hw-event-bmc-secret.yaml` file:

```
apiVersion: v1
kind: Secret
metadata:
  name: redfish-basic-auth
type: Opaque
stringData: ①
  username: <bmc_username>
  password: <bmc_password>
  # BMC host DNS or IP address
  hostaddr: <bmc_host_ip_address>
```

① Enter plain text values for the various items under `stringData`.

b. Create the `Secret` CR:

```
$ oc create -f hw-event-bmc-secret.yaml
```

Subscribing applications to Redfish hardware event notifications REST API reference

Use the Redfish event notifications REST API to subscribe an application to the Redfish hardware events that are generated on the parent node.

Subscribe applications to Redfish events by using the resource address `/cluster/node/<node_name>/redfish/event`, where `<node_name>` is the cluster node running the application.

Deploy your `cloud-event-consumer` application container and `cloud-event-proxy` sidecar container in a separate application pod. The `cloud-event-consumer` application subscribes to the `cloud-event-proxy` container in the application pod.

Use the following API endpoints to subscribe the `cloud-event-consumer` application to Redfish events posted by the `cloud-event-proxy` container at `http://localhost:8089/api/cloudNotifications/v1/` in the application pod:

- `/api/cloudNotifications/v1/subscriptions`
  - `POST`: Creates a new subscription
  - `GET`: Retrieves a list of subscriptions
- `/api/cloudNotifications/v1/subscriptions/<subscription_id>`
  - `GET`: Returns details for the specified subscription ID
- `api/cloudNotifications/v1/subscriptions/status/<subscription_id>`
  - `PUT`: Creates a new status ping request for the specified subscription ID
- `/api/cloudNotifications/v1/health`

- `GET`: Returns the health status of `cloudNotifications` API

`9089` is the default port for the `cloud-event-consumer` container deployed in the application pod. You can configure a different port for your application as required.

api/cloudNotifications/v1/subscriptions

HTTP method

`GET api/cloudNotifications/v1/subscriptions`

Description

Returns a list of subscriptions. If subscriptions exist, a `200 OK` status code is returned along with the list of subscriptions.

Example API response

```
[
 {
  "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri": "http://localhost:9089/api/cloudNotifications/v1/dummy",
  "uriLocation":
"http://localhost:8089/api/cloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-
8d3a-666c24e34d32",
  "resource": "/cluster/node/openshift-worker-
0.openshift.example.com/redfish/event"
 }
]
```

HTTP method

`POST api/cloudNotifications/v1/subscriptions`

Description

Creates a new subscription. If a subscription is successfully created, or if it already exists, a `201 Created` status code is returned.

Table 3. Query parameters

Parameter	Type
subscription	data

Example payload

```
{
  "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/openshift-worker-
0.openshift.example.com/redfish/event"
}
```

api/cloudNotifications/v1/subscriptions/<subscription_id>

HTTP method

`GET api/cloudNotifications/v1/subscriptions/<subscription_id>`

Description

Returns details for the subscription with ID `<subscription_id>`

Table 4. Query parameters

Parameter	Type
`<subscription_id>`	string

Example API response

```
{
  "id":"ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri":"http://localhost:9089/api/cloudNotifications/v1/dummy",
  "uriLocation"
:"http://localhost:8089/api/cloudNotifications/v1/subscriptions/ca11ab76-86f9-
428c-8d3a-666c24e34d32",
  "resource":"/cluster/node/openshift-worker-
0.openshift.example.com/redfish/event"
}
```

api/cloudNotifications/v1/subscriptions/status/<subscription_id>

HTTP method

`PUT api/cloudNotifications/v1/subscriptions/status/<subscription_id>`

Description

Creates a new status ping request for subscription with ID `<subscription_id>`. If a subscription is present, the status request is successful and a `202 Accepted` status code is returned.

Table 5. Query parameters

Parameter	Type
`<subscription_id>`	string

Example API response

```
{"status":"ping sent"}
```

api/cloudNotifications/v1/health/

HTTP method

`GET api/cloudNotifications/v1/health/`

Description

Returns the health status for the `cloudNotifications` REST API.

Example API response

```
OK
```

Cluster Group Upgrades Operator for cluster updates

You can use the Cluster Group Upgrades Operator (CGUO) to manage the software lifecycle of multiple single-node OpenShift clusters. The CGUO uses Red Hat Advanced Cluster Management (RHACM) policies to perform changes on the target clusters.

The Cluster Group Upgrades Operator is a Developer Preview feature only. Developer Preview features are not supported with Red Hat production service level agreements (SLAs) and are not functionally complete or production-ready. Do not use Developer Preview features for production or business-critical workloads. Developer Preview features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. These features might not have any documentation, and testing is limited. Red Hat might provide ways to submit feedback on Developer Preview features without an associated SLA.

About the Cluster Group Upgrades Operator configuration

The Cluster Group Upgrades Operator (CGUO) manages the deployment of Red Hat Advanced Cluster Management (RHACM) policies for one or more OpenShift Container Platform clusters. Using the CGUO in a large network of clusters allows the phased rollout of policies to the clusters in limited batches. This helps to minimize possible service disruptions when updating. With the CGUO, you can control the following actions:

- The timing of the update
- The number of RHACM-managed clusters
- The subset of managed clusters to apply the policies to
- The update order of the clusters
- The set of policies remediated to the cluster
- The order of policies remediated to the cluster

The CGUO supports the orchestration of the OpenShift Container Platform y-stream and z-stream updates, and day-two operations on y-streams and z-streams.

About managed policies used with Cluster Group Upgrades Operator

The Cluster Group Upgrades Operator (CGUO) uses RHACM policies for cluster updates.

The CGUO can be used to manage the rollout of any policy CR where the `remediationAction` field is set to `inform`. Supported use cases include the following:

- Manual user creation of policy CRs
- Automatically generated policies from the `PolicyGenTemplate` custom resource definition (CRD)

For policies that update an Operator subscription with manual approval, the CGUO provides additional functionality that approves the installation of the updated Operator.

For more information about managed policies, see Policy Overview in the RHACM documentation.

For more information about the `PolicyGenTemplate` CRD, see the "About the PolicyGenTemplate" section in "Deploying distributed units at scale in a disconnected environment".

Installing the upstream Cluster Group Upgrades Operator

The Cluster Group Upgrades Operator is not available in the `redhat-operators` Operators source for OpenShift Container Platform version 4.10. To install the Operator, see Installing upstream RAN operators.

The Cluster Group Upgrades Operator is not supported with Red Hat production service level agreements (SLAs) and is not functionally complete or production-ready. The Operator is provided for preview purposes only. Do not use Cluster Group Upgrades Operator (CGUO) for production or business-critical workloads.

About the ClusterGroupUpgrade CR

The Cluster Group Upgrades Operator (CGUO) builds the remediation plan from the `ClusterGroupUpgrade` CR for a group of clusters. You can define the following specifications in a `ClusterGroupUpgrade` CR:

- Clusters in the group
- Blocking `ClusterGroupUpgrade` CRs
- Applicable list of managed policies
- Number of concurrent updates
- Applicable canary updates
- Actions to perform before and after the update
- Update timing

As the CGUO works through remediation of the policies to the specified clusters, the `ClusterGroupUpgrade` CR can have the following states:

- `UpgradeNotStarted`

- `UpgradeCannotStart`
- `UpgradeNotComplete`
- `UpgradeTimedOut`
- `UpgradeCompleted`
- `PrecachingRequired`

After the CGUO completes a cluster update, the cluster does not update again under the control of the same `ClusterGroupUpgrade` CR. You must create a new `ClusterGroupUpgrade` CR in the following cases:

- When you need to update the cluster again
- When the cluster changes to non-compliant with the `inform` policy after being updated

The UpgradeNotStarted state

The initial state of the `ClusterGroupUpgrade` CR is `UpgradeNotStarted`.

The CGUO builds a remediation plan based on the following fields:

- The `clusterSelector` field specifies the labels of the clusters that you want to update.
- The `clusters` field specifies a list of clusters to update.
- The `canaries` field specifies the clusters for canary updates.
- The `maxConcurrency` field specifies the number of clusters to update in a batch.

You can use the `clusters` and the `clusterSelector` fields together to create a combined list of clusters.

The remediation plan starts with the clusters listed in the `canaries` field. Each canary cluster forms a single-cluster batch.

Any failures during the update of a canary cluster stops the update process.

The `ClusterGroupUpgrade` CR transitions to the `UpgradeNotCompleted` state after the remediation plan is successfully created and after the `enable` field is set to `true`. At this point, the CGUO starts to update the non-compliant clusters with the specified managed policies.

You can only make changes to the `spec` fields if the `ClusterGroupUpgrade` CR is either in the `UpgradeNotStarted` or the `UpgradeCannotStart` state.

Sample `ClusterGroupUpgrade` CR in the `UpgradeNotStarted` state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters: ①
  - spoke1
  enable: false
  managedPolicies: ②
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy: ③
    canaries: ④
      - spoke1
    maxConcurrency: 1 ⑤
    timeout: 240
status: ⑥
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  placementBindings:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  placementRules:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  remediationPlan:
  - - spoke1
```

① Defines the list of clusters to update.

② Lists the user-defined set of policies to remediate.

③ Defines the specifics of the cluster updates.

④ Defines the clusters for canary updates.

⑤ Defines the maximum number of concurrent updates in a batch. The number of remediation batches is the number of canary clusters, plus the number of clusters, except the canary clusters, divided by the `maxConcurrency` value. The clusters that are already compliant with all

the managed policies are excluded from the remediation plan.

⑥ Displays information about the status of the updates.

The UpgradeCannotStart state

In the `UpgradeCannotStart` state, the update cannot start because of the following reasons:

- Blocking CRs are missing from the system
- Blocking CRs have not yet finished

The UpgradeNotCompleted state

In the `UpgradeNotCompleted` state, the CGUO enforces the policies following the remediation plan defined in the `UpgradeNotStarted` state.

Enforcing the policies for subsequent batches starts immediately after all the clusters of the current batch are compliant with all the managed policies. If the batch times out, the CGUO moves on to the next batch. The timeout value of a batch is the `spec.timeout` field divided by the number of batches in the remediation plan.

The managed policies apply in the order that they are listed in the `managedPolicies` field in the `ClusterGroupUpgrade` CR. One managed policy is applied to the specified clusters at a time. After the specified clusters comply with the current policy, the next managed policy is applied to the next non-compliant cluster.

Sample `ClusterGroupUpgrade` CR in the `UpgradeNotCompleted` state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true ①
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: ②
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non
compliant
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  placementBindings:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  placementRules:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  remediationPlan:
  - - spoke1
  status:
    currentBatch: 1
    remediationPlanForBatch: ③
      spoke1: 0
```

① The update starts when the value of the `spec.enable` field is `true`.

② The `status` fields change accordingly when the update begins.

③ Lists the clusters in the batch and the index of the policy that is being currently applied to each cluster. The index of the policies starts with `0` and the index follows the order of the `status.managedPoliciesForUpgrade` list.

The UpgradeTimedOut state

In the `UpgradeTimedOut` state, the CGUO checks every hour if all the policies for the `ClusterGroupUpgrade` CR are compliant. The checks continue until the `ClusterGroupUpgrade` CR is deleted or the updates are completed. The periodic checks allow the updates to complete if they get prolonged due to network, CPU, or other issues.

The CGUO transitions to the `UpgradeTimedOut` state in two cases:

- When the current batch contains canary updates and the cluster in the batch does not comply with all the managed policies within the batch timeout.
- When the clusters do not comply with the managed policies within the `timeout` value specified in the `remediationStrategy` field.

If the policies are compliant, the CGUO transitions to the `UpgradeCompleted` state.

The UpgradeCompleted state

In the `UpgradeCompleted` state, the cluster updates are complete.

Sample `ClusterGroupUpgrade` CR in the `UpgradeCompleted` state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  actions:
    afterCompletion:
      deleteObjects: true ①
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: ②
  conditions:
  - message: The ClusterGroupUpgrade CR has all clusters compliant with all the
managed policies
    reason: UpgradeCompleted
    status: "True"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  remediationPlan:
  - - spoke1
  status:
    remediationPlanForBatch:
      spoke1: -2 ③
```

① The value of `spec.action.afterCompletion.deleteObjects` field is `true` by default. After the update is completed, the CGUO deletes the underlying RHACM objects that were created during the update. This option is to prevent the RHACM hub from continuously checking for compliance after a successful update.

② The `status` fields show that the updates completed successfully.

③ Displays that all the policies are applied to the cluster.

The PrecachingRequired state

In the `PrecachingRequired` state, the clusters need to have images pre-cached before the update can start. For more information about pre-caching, see the "Using the container image pre-

cache feature" section.

Blocking ClusterGroupUpgrade CRs

You can create multiple `ClusterGroupUpgrade` CRs and control their order of application.

For example, if you create `ClusterGroupUpgrade` CR C that blocks the start of `ClusterGroupUpgrade` CR A, then `ClusterGroupUpgrade` CR A cannot start until the status of `ClusterGroupUpgrade` CR C becomes `UpgradeComplete`.

One `ClusterGroupUpgrade` CR can have multiple blocking CRs. In this case, all the blocking CRs must complete before the upgrade for the current CR can start.

Prerequisites

- Install the Cluster Group Upgrades Operator (CGUO).
- Provision one or more managed clusters.
- Log in as a user with `cluster-admin` privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Save the content of the `ClusterGroupUpgrade` CRs in the `cgu-a.yaml`, `cgu-b.yaml`, and `cgu-c.yaml` files.

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs: ①
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
```

```
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
```

① Defines the blocking CRs. The `cgu-a` update cannot start until `cgu-c` is complete.

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs: ①
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

```
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

① The `cgu-b` update cannot start until `cgu-a` is complete.

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec: ①
  clusters:
  - spoke6
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status: {}
```

① The `cgu-c` update does not have any blocking CRs. The CGUO starts the `cgu-c` update when the `enable` field is set to `true`.

2. Create the `ClusterGroupUpgrade` CRs by running the following command for each relevant CR:

```
$ oc apply -f <name>.yaml
```

3. Start the update process by running the following command for each relevant CR:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/<name> \
--type merge -p '{"spec":{"enable":true}}'
```

The following examples show `ClusterGroupUpgrade` CRs where the `enable` field is set to `true`:

Example for `cgu-a` with blocking CRs

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs:
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not
yet
      completed: [cgu-c]' ①
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
```

```
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
  status: {}
```

① Shows the list of blocking CRs.

Example for `cgu-b` with blocking CRs

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs:
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not
yet
      completed: [cgu-a]' ①
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  copiedPolicies:
```

```
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

① Shows the list of blocking CRs.

Example for cgu-c with blocking CRs

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec:
  clusters:
  - spoke6
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non
compliant ①
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status:
    currentBatch: 1
    remediationPlanForBatch:
      spoke6: 0
```

① The cgu-c update does not have any blocking CRs.

Update policies on managed clusters

The Cluster Group Upgrades Operator (CGUO) remediates a set of `inform` policies for the clusters specified in the `ClusterGroupUpgrade` CR. The CGUO remediates `inform` policies by making `enforce` copies of the managed RHACM policies. Each copied policy has its own corresponding RHACM placement rule and RHACM placement binding.

One by one, the CGUO adds each cluster from the current batch to the placement rule that corresponds with the applicable managed policy. If a cluster is already compliant with a policy, the CGUO skips applying that policy on the compliant cluster. The CGUO then moves on to applying the next policy to the non-compliant cluster. After the CGUO completes the updates in a batch, all clusters are removed from the placement rules associated with the copied policies. Then, the update of the next batch starts.

If a spoke cluster does not report any compliant state to RHACM, the managed policies on the hub cluster can be missing status information that the CGUO needs. The CGUO handles these cases in the following ways:

- If a policy's `status.compliant` field is missing, the CGUO ignores the policy and the CGUO adds a log entry. Then the CGUO continues looking at the policy's `status.status` field.
- If a policy's `status.status` is missing, the CGUO produces an error.
- If a cluster's compliance status is missing in the policy's `status.status` field, the CGUO considers that cluster to be non-compliant with that policy.

For more information about RHACM policies, see Policy overview.

Additional resources

For more information about `PolicyGenTemplate` CRD, see About the PolicyGenTemplate.

Applying update policies to managed clusters

You can update your managed clusters by applying your policies.

Prerequisites

- Install the Cluster Group Upgrades Operator (CGUO).
- Provision one or more managed clusters.
- Log in as a user with `cluster-admin` privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Save the contents of the `ClusterGroupUpgrade` CR in the `cgu-1.yaml` file.

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-1
  namespace: default
spec:
  managedPolicies: ①
    - policy1-common-cluster-version-policy
    - policy2-common-pao-sub-policy
    - policy3-common-ptp-sub-policy
    - policy4-common-sriov-sub-policy
  enable: false
  clusters: ②
  - spoke1
  - spoke2
  - spoke5
  - spoke6
  remediationStrategy:
    maxConcurrency: 2 ③
    timeout: 240 ④
```

① The name of the policies to apply.

② The list of clusters to update.

③ The `maxConcurrency` field signifies the number of clusters updated at the same time.

④ The update timeout in minutes.

2. Create the `ClusterGroupUpgrade` CR by running the following command:

```
$ oc create -f cgu-1.yaml
```

   a. Check if the `ClusterGroupUpgrade` CR was created in the hub cluster by running the following command:

```
$ oc get cgu --all-namespaces
```

   Example output

```
NAMESPACE   NAME      AGE
default     cgu-1     8m55s
```

   b. Check the status of the update by running the following command:

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

Example output


```
{
  "computedMaxConcurrency": 2,
  "conditions": [
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "The ClusterGroupUpgrade CR is not enabled", ①
      "reason": "UpgradeNotStarted",
      "status": "False",
      "type": "Ready"
    }
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-pao-sub-policy": "[{\"kind\":\"Subscription\",\"name\"
:\"performance-addon-operator\",\"namespace\":\"openshift-performance-
addon-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\"
:\"ptp-operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"
name\":\"sriov-network-operator-subscription\",\"namespace\":\"openshift-
sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-pao-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-pao-sub-policy": "default",
```

```
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "precaching": {
    "spec": {}
  },
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {}
}
```

① The `spec.enable` field in the `ClusterGroupUpgrade` CR is set to `false`.

c. Check the status of the policies by running the following command:

```
$ oc get policies -A
```

Example output

```
NAMESPACE   NAME
REMEDIATION ACTION   COMPLIANCE STATE   AGE
default     cgu-policy1-common-cluster-version-policy           enforce
17m ①
default     cgu-policy2-common-pao-sub-policy                   enforce
17m
default     cgu-policy3-common-ptp-sub-policy                   enforce
17m
default     cgu-policy4-common-sriov-sub-policy                 enforce
17m
default     policy1-common-cluster-version-policy               inform
NonCompliant       15h
default     policy2-common-pao-sub-policy                       inform
NonCompliant       15h
default     policy3-common-ptp-sub-policy                       inform
NonCompliant       18m
default     policy4-common-sriov-sub-policy                     inform
NonCompliant       18m
```

① The `spec.remediationAction` field of policies currently applied on the clusters is set to `enforce`. The managed policies in `inform` mode from the `ClusterGroupUpgrade` CR remain in `inform` mode during the update.

3. Change the value of the `spec.enable` field to `true` by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-1 \
--patch '{"spec":{"enable":true}}' --type=merge
```

Verification

1. Check the status of the update again by running the following command:

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

Example output

```
{
  "computedMaxConcurrency": 2,
  "conditions": [ ①
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "The ClusterGroupUpgrade CR has upgrade policies that are
still non compliant",
      "reason": "UpgradeNotCompleted",
      "status": "False",
      "type": "Ready"
    }
```

```
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-pao-sub-policy": "[{\"kind\":\"Subscription\",\"name\":
\"performance-addon-operator\",\"namespace\":\"openshift-performance-addon-
operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":
\"ptp-operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":
\"sriov-network-operator-subscription\",\"namespace\":\"openshift-sriov-
network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-pao-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-pao-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
```

```
      "cgu-policy3-common-ptp-sub-policy",
      "cgu-policy4-common-sriov-sub-policy"
    ],
    "precaching": {
      "spec": {}
    },
    "remediationPlan": [
      [
        "spoke1",
        "spoke2"
      ],
      [
        "spoke5",
        "spoke6"
      ]
    ],
    "status": {
      "currentBatch": 1,
      "currentBatchStartedAt": "2022-02-25T15:54:16Z",
      "remediationPlanForBatch": {
        "spoke1": 0,
        "spoke2": 1
      },
      "startedAt": "2022-02-25T15:54:16Z"
    }
}
```

① Reflects the update progress of the current batch. Run this command again to receive updated information about the progress.

2. If the policies include Operator subscriptions, you can check the installation progress directly on the single-node cluster.

   a. Export the `KUBECONFIG` file of the single-node cluster you want to check the installation progress for by running the following command:

   ```
   $ export KUBECONFIG=<cluster_kubeconfig_absolute_path>
   ```

   b. Check all the subscriptions present on the single-node cluster and look for the one in the policy you are trying to install through the `ClusterGroupUpgrade` CR by running the following command:

   ```
   $ oc get subs -A | grep -i <subscription_name>
   ```

Example output for `cluster-logging` policy

```
NAMESPACE                              NAME                          PACKAGE
SOURCE             CHANNEL
openshift-logging                      cluster-logging
cluster-logging             redhat-operators   stable
```

3. If one of the managed policies includes a `ClusterVersion` CR, check the status of platform updates in the current batch by running the following command against the spoke cluster:

```
$ oc get clusterversion
```

Example output

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.9.5     True        True          43s     Working towards 4.9.7: 71
of 735 done (9% complete)
```

4. Check the Operator subscription by running the following command:

```
$ oc get subs -n <operator-namespace> <operator-subscription> -ojsonpath
="{.status}"
```

5. Check the install plans present on the single-node cluster that is associated with the desired subscription by running the following command:

```
$ oc get installplan -n <subscription_namespace>
```

Example output for `cluster-logging` Operator

```
NAMESPACE                              NAME            CSV
APPROVAL   APPROVED
openshift-logging                      install-6khtw   cluster-logging.5.3.3-4
Manual     true ①
```

① The install plans have their `Approval` field set to `Manual` and their `Approved` field changes from `true` to `false` after the CGUO approves the install plan.

6. Check if the cluster service version for the Operator of the policy that the `ClusterGroupUpgrade` is installing reached the `Succeeded` phase by running the following command:

```
$ oc get csv -n <operator_namespace>
```

Example output for OpenShift Logging Operator

```
NAMESPACE                              NAME            DISPLAY          VERSION
REPLACES   PHASE
openshift-operator-lifecycle-manager   packageserver   Package Server   0.18.3
Succeeded
```

Using the container image pre-cache feature

Clusters might have limited bandwidth to access the container image registry, which can cause a timeout before the updates are completed.

The time of the update is not set by the CGUO. You can apply the `ClusterGroupUpgrade` CR at the beginning of the update by manual application or by external automation.

The container image pre-caching starts when the `preCaching` field is set to `true` in the `ClusterGroupUpgrade` CR. After a successful pre-caching process, you can start remediating policies. The remediation actions start when the `enable` field is set to `true`.

The pre-caching process can be in the following statuses:

`PrecacheNotStarted`

This is the initial state all clusters are automatically assigned to on the first reconciliation pass of the `ClusterGroupUpgrade` CR.

In this state, the CGUO deletes any pre-caching namespace and hub view resources of spoke clusters that remain from previous incomplete updates. The CGUO then creates a new `ManagedClusterView` resource for the spoke pre-caching namespace to verify its deletion in the `PrecachePreparing` state.

`PrecachePreparing`

Cleaning up any remaining resources from previous incomplete updates is in progress.

`PrecacheStarting`

Pre-caching job prerequisites and the job are created.

`PrecacheActive`

The job is in "Active" state.

`PrecacheSucceeded`

The pre-cache job has succeeded.

`PrecacheTimeout`

The artifact pre-caching has been partially done.

`PrecacheUnrecoverableError`

The job ends with a non-zero exit code.

Creating a ClusterGroupUpgrade CR with pre-caching

The pre-cache feature allows the required container images to be present on the spoke cluster before the update starts.

Prerequisites

- Install the Cluster Group Upgrades Operator (CGUO).
- Provision one or more managed clusters.
- Log in as a user with `cluster-admin` privileges.

Procedure

1. Save the contents of the `ClusterGroupUpgrade` CR with the `preCaching` field set to `true` in the `clustergroupupgrades-group-du.yaml` file:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true ①
  clusters:
  - cnfdb1
  - cnfdb2
  enable: false
  managedPolicies:
  - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

① The `preCaching` field is set to `true`, which enables the CGUO to pull the container images before starting the update.

2. When you want to start the update, apply the `ClusterGroupUpgrade` CR by running the following command:

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

Verification

1. Check if the `ClusterGroupUpgrade` CR exists in the hub cluster by running the following command:

```
$ oc get cgu -A
```

Example output

```
NAMESPACE          NAME              AGE
ztp-group-du-sno   du-upgrade-4918   10s ①
```

① The CR is created.

2. Check the status of the pre-caching task by running the following command:

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

Example output

```
{
  "conditions": [
    {
      "lastTransitionTime": "2022-01-27T19:07:24Z",
      "message": "Precaching is not completed (required)", ①
      "reason": "PrecachingRequired",
      "status": "False",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:24Z",
      "message": "Precaching is required and not done",
      "reason": "PrecachingNotDone",
      "status": "False",
      "type": "PrecachingDone"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:34Z",
      "message": "Pre-caching spec is valid and consistent",
      "reason": "PrecacheSpecIsWellFormed",
      "status": "True",
      "type": "PrecacheSpecValid"
    }
  ],
  "precaching": {
    "clusters": [
      "cnfdb1" ②
    ],
    "spec": {
      "platformImage": "image.example.io"},
    "status": {
      "cnfdb1": "Active"}
    }
}
```

① Displays that the update is in progress.

② Displays the list of identified clusters.

3. Check the status of the pre-caching job by running the following command on the spoke cluster:

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

Example output

```
NAME                  COMPLETIONS   DURATION   AGE
job.batch/pre-cache   0/1           3m10s      3m10s

NAME                      READY   STATUS    RESTARTS   AGE
pod/pre-cache--1-9bmlr    1/1     Running   0          3m10s
```

4. Check the status of the `ClusterGroupUpgrade` CR by running the following command:

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

Example output

```
"conditions": [
    {
      "lastTransitionTime": "2022-01-27T19:30:41Z",
      "message": "The ClusterGroupUpgrade CR has all clusters compliant with
all the managed policies",
      "reason": "UpgradeCompleted",
      "status": "True",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-01-27T19:28:57Z",
      "message": "Precaching is completed",
      "reason": "PrecachingCompleted",
      "status": "True",
      "type": "PrecachingDone" ①
    }
```

① The pre-cache tasks are done.

Troubleshooting the Cluster Group Upgrades Operator

The Cluster Group Upgrades Operator (CGUO) is an OpenShift Container Platform Operator that remediates RHACM policies. When issues occur, use the `oc adm must-gather` command to gather details and logs and to take steps in debugging the issues.

For more information about related topics, see the following documentation:

- Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix
- Red Hat Advanced Cluster Management Troubleshooting
- The "Troubleshooting Operator issues" section

General troubleshooting

You can determine the cause of the problem by reviewing the following questions:

- Is the configuration that you are applying supported?
  - Are the RHACM and the OpenShift Container Platform versions compatible?
  - Are the CGUO and RHACM versions compatible?
- Which of the following components is causing the problem?
  - Managed policies
  - Clusters
  - Remediation Strategy
  - Cluster Group Upgrades Operator

To ensure that the `ClusterGroupUpgrade` configuration is functional, you can do the following:

1. Create the `ClusterGroupUpgrade` CR with the `spec.enable` field set to `false`.
2. Wait for the status to be updated and go through the troubleshooting questions.
3. If everything looks as expected, set the `spec.enable` field to `true` in the `ClusterGroupUpgrade` CR.

After you set the `spec.enable` field to `true` in the `ClusterUpgradeGroup` CR , the update procedure starts and you cannot edit the CR's `spec` fields anymore.

Cannot modify the ClusterUpgradeGroup CR

Issue

You cannot edit the `ClusterUpgradeGroup` CR after enabling the update.

Resolution

Restart the procedure by performing the following steps:

1. Remove the old `ClusterGroupUpgrade` CR by running the following command:

```
$ oc delete cgu -n <ClusterGroupUpgradeCR_namespace>
<ClusterGroupUpgradeCR_name>
```

2. Check and fix the existing issues with the managed clusters and policies.
   a. Ensure that all the clusters are managed clusters and available.
   b. Ensure that all the policies exist and have the `spec.remediationAction` field set to `inform`.
3. Create a new `ClusterGroupUpgrade` CR with the correct configurations.

```
$ oc apply -f <ClusterGroupUpgradeCR_YAML>
```

Managed policies

Checking managed policies on the system

Issue

You want to check if you have the correct managed policies on the system.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.managedPolicies}'
```

Example output

```
["group-du-sno-validator-du-validator-policy", "policy2-common-pao-sub-policy",
"policy3-common-ptp-sub-policy"]
```

Checking remediationAction mode

Issue

You want to check if the `remediationAction` field is set to `inform` in the `spec` of the managed policies.

Resolution

Run the following command:

```
$ oc get policies --all-namespaces
```

Example output

```
NAMESPACE   NAME                                        REMEDIATION
ACTION   COMPLIANCE STATE   AGE
default     policy1-common-cluster-version-policy       inform
NonCompliant       5d21h
default     policy2-common-pao-sub-policy               inform
Compliant          5d21h
default     policy3-common-ptp-sub-policy               inform
NonCompliant       5d21h
default     policy4-common-sriov-sub-policy             inform
NonCompliant       5d21h
```

Checking policy compliance state

Issue

You want to check the compliance state of policies.

Resolution

Run the following command:

```
$ oc get policies --all-namespaces
```

Example output

```
NAMESPACE   NAME                                        REMEDIATION
ACTION   COMPLIANCE STATE   AGE
default     policy1-common-cluster-version-policy       inform
NonCompliant       5d21h
default     policy2-common-pao-sub-policy               inform
Compliant          5d21h
default     policy3-common-ptp-sub-policy               inform
NonCompliant       5d21h
default     policy4-common-sriov-sub-policy             inform
NonCompliant       5d21h
```

Clusters

Checking if managed clusters are present

Issue

You want to check if the clusters in the `ClusterGroupUpgrade` CR are managed clusters.

Resolution

Run the following command:

```
$ oc get managedclusters
```

Example output

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED
AVAILABLE   AGE
local-cluster   true           https://api.hub.example.com:6443      True
Unknown     13d
spoke1          true           https://api.spoke1.example.com:6443   True
True        13d
spoke3          true           https://api.spoke3.example.com:6443   True
True        27h
```

1. Alternatively, check the CGUO manager logs:
   a. Get the name of the CGUO manager by running the following command:

```
$ oc get pod -n openshift-cluster-group-upgrades
```

Example output

```
NAME                                                     READY
STATUS    RESTARTS   AGE
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp   2/2
Running   0          45m
```

   b. Check the CGUO manager logs by running the following command:

```
$ oc logs -n openshift-cluster-group-upgrades \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

Example output

```
ERROR   controller-runtime.manager.controller.clustergroupupgrade
Reconciler error    {"reconciler group": "ran.openshift.io", "reconciler
kind": "ClusterGroupUpgrade", "name": "lab-upgrade", "namespace":
"default", "error": "Cluster spoke5555 is not a ManagedCluster"} ①
sigs.k8s.io/controller-
runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

① The error message shows that the cluster is not a managed cluster.

Checking if managed clusters are available

Issue

You want to check if the managed clusters specified in the `ClusterGroupUpgrade` CR are available.

Resolution

Run the following command:

```
$ oc get managedclusters
```

Example output

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                    JOINED
AVAILABLE   AGE
local-cluster   true           https://api.hub.testlab.com:6443        True
Unknown     13d
spoke1          true           https://api.spoke1.testlab.com:6443     True
True        13d ①
spoke3          true           https://api.spoke3.testlab.com:6443     True
True        27h ①
```

① The value of the `AVAILABLE` field is `True` for the managed clusters.

Checking clusterSelector

Issue

You want to check if the `clusterSelector` field is specified in the `ClusterGroupUpgrade` CR in at least one of the managed clusters.

Resolution

Run the following command:

```
$ oc get managedcluster --selector=upgrade=true ①
```

① The label for the clusters you want to update is `upgrade:true`.

Example output

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                    JOINED
AVAILABLE   AGE
spoke1          true           https://api.spoke1.testlab.com:6443     True
True        13d
spoke3          true           https://api.spoke3.testlab.com:6443     True
True        27h
```

Checking if canary clusters are present

Issue

You want to check if the canary clusters are present in the list of clusters.

Example `ClusterGroupUpgrade` *CR*

```
spec:
    clusters:
    - spoke1
    - spoke3
    clusterSelector:
    - upgrade2=true
    remediationStrategy:
        canaries:
        - spoke3
        maxConcurrency: 2
        timeout: 240
```

Resolution

Run the following commands:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.clusters}'
```

Example output

```
["spoke1", "spoke3"]
```

1. Check if the canary clusters are present in the list of clusters that match `clusterSelector` labels by running the following command:

```
$ oc get managedcluster --selector=upgrade=true
```

Example output

```
NAME           HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED    AVAILABLE
AGE
spoke1         true           https://api.spoke1.testlab.com:6443   True
True        13d
spoke3         true           https://api.spoke3.testlab.com:6443   True
True        27h
```

A cluster can be present in `spec.clusters` and also be matched by the `spec.clusterSelecter` label.

Checking the pre-caching status on spoke clusters

1. Check the status of pre-caching by running the following command on the spoke cluster:

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

Remediation Strategy

Checking if remediationStrategy is present in the ClusterGroupUpgrade CR

Issue

You want to check if the `remediationStrategy` is present in the `ClusterGroupUpgrade` CR.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy}'
```

Example output

```
{"maxConcurrency":2, "timeout":240}
```

Checking if maxConcurrency is specified in the ClusterGroupUpgrade CR

Issue

You want to check if the `maxConcurrency` is specified in the `ClusterGroupUpgrade` CR.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath
='{.spec.remediationStrategy.maxConcurrency}'
```

Example output

```
2
```

Cluster Group Upgrades Operator

Checking condition message and status in the ClusterGroupUpgrade CR

Issue

You want to check the value of the `status.conditions` field in the `ClusterGroupUpgrade` CR.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.conditions}'
```

Example output

```
{"lastTransitionTime":"2022-02-17T22:25:28Z", "message":"The
ClusterGroupUpgrade CR has managed policies that are
missing:[policyThatDoesntExist]", "reason":"UpgradeCannotStart", "status"
:"False", "type":"Ready"}
```

Checking corresponding copied policies

Issue

You want to check if every policy from `status.managedPoliciesForUpgrade` has a corresponding policy in `status.copiedPolicies`.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -oyaml
```

Example output

```
status:
  ...
  copiedPolicies:
  - lab-upgrade-policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy3-common-ptp-sub-policy
    namespace: default
```

Checking if status.remediationPlan was computed

Issue

You want to check if `status.remediationPlan` is computed.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.remediationPlan}'
```

Example output

```
[["spoke2", "spoke3"]]
```

Errors in the CGUO manager container

Issue

You want to check the logs of the manager container of the CGUO.

Resolution

Run the following command:

```
$ oc logs -n openshift-cluster-group-upgrades \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

Example output

```
ERROR   controller-runtime.manager.controller.clustergroupupgrade   Reconciler
error    {"reconciler group": "ran.openshift.io", "reconciler kind":
"ClusterGroupUpgrade", "name": "lab-upgrade", "namespace": "default", "error":
"Cluster spoke5555 is not a ManagedCluster"} ①
sigs.k8s.io/controller-
runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

① Displays the error.

Additional resources

- For information about troubleshooting, see OpenShift Container Platform Troubleshooting Operator Issues.
- For more information about using Cluster Group Upgrades Operator in the ZTP workflow, see Updating managed policies with the CGUO.

Deploying distributed units manually on single node OpenShift

The procedures in this topic tell you how to manually deploy clusters on a small number of single nodes as a distributed unit (DU) during installation.

The procedures do not describe how to install single node OpenShift (SNO). This can be accomplished through many mechanisms. Rather, they are intended to capture the elements that should be configured as part of the installation process:

- Networking is needed to enable connectivity to the SNO DU when the installation is complete.
- Workload partitioning, which can only be configured during installation.
- Additional items that help minimize the potential reboots post installation.

Configuring the distributed units (DUs)

This section describes a set of configurations for an OpenShift Container Platform cluster so that it meets the feature and performance requirements necessary for running a distributed unit (DU) application. Some of this content must be applied during installation and other configurations can be applied post-install.

After you have installed the single-node DU, further configuration is needed to enable the platform to carry a DU workload.

The configurations in this section are applied to the cluster after installation in order to configure the cluster for DU workloads.

Enabling workload partitioning

A key feature to enable as part of a single node installation is workload partitioning. This limits the cores allowed to run platform services, maximizing the CPU core for application payloads. You must configure workload partitioning at cluster installation time.

Workload partitioning must be applied during installation.

Procedure

- The base64-encoded content below contains the CPU set that the management workloads are constrained to. This content must be adjusted to match the set specified in the `performanceprofile` and must be accurate for the number of cores on the cluster.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 02-master-workload-partitioning
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl9hbm5vdG
F0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVudCIKYW5ub3RhdGlvb
l9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5pbyIKcmVzb3VyY2VzID0geyAi
Y3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMSw1Mi01MyIgfQo=
        mode: 420
        overwrite: true
        path: /etc/crio/crio.conf.d/01-workload-partitioning
        user:
          name: root
      - contents:
          source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTEsNTItNTMiCiAgfQp9Cg
==
        mode: 420
        overwrite: true
        path: /etc/kubernetes/openshift-workload-pinning
        user:
          name: root
```

Configuring the container mount namespace

To reduce the overall management footprint of the platform, a machine configuration is provided to contain the mount points. No configuration changes are needed. Use the provided settings:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: container-mount-namespace-and-kubelet-conf-master
spec:
  config:
    ignition:
```

```
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKCmRlYnVnKCkgewogIGVjaG8gJEAgPiYyCn0KCnVzYWdlKCkgewogIGVja
G8gVXNhZ2U6ICQoYmFzZW5hbWUgJDApIFVOSVQgW2VudmZpbGUgW3Zhcm5hbWVdXQogIGVjaG8KICBlY2h
vIEV4dHJhY3QgdGhlIGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeGVjU3RhcnQgc3Rhbnphc IGZyb20gdGhlI
GdpdmVuIHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJuIGl0IHRvIHN0ZG91dAogIGVjaG8KICBlY2hvICJJZiA
nZW52ZmlsZScgaXMgcHJvdmlkZWQsIHB1dCBpdCBpbiB0aGVyZSBpbnN0ZWFkLCBhcyBhbiBlbnZpcm9ub
WVudCB2YXJpYWJsZSBuYW1lZCAndmFybmFtZSciCiAgZWNobyAiRGVmYXVsdCAndmFybmFtZScgaXMgRVh
FQ1NUQVJUIGlmIG5vdCBzcGVjaWZpZWQiCiAgZXhpdCAxCn0KClVOSVQ9JDEKRU5WRklMRT0kMgpWQVJOQ
U1FPSQzCmlmIFtbIC16ICRVTklUIHx8ICRVTklUID09ICItLWhlbHAiIHx8ICRVTklUID09ICItaCIgXV0
7IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cmFjdGluZyBFeGVjU3RhcnQgZnJvbSAkVU5JVCIKRklMR
T0kKHN5c3RlbWN0bCBjYXQgJFVOSVQgfCBoZWFkIC1uIDEpCkZJTEU9JHtGSUxFI1wjIH0KaWYgW1sgISA
tZiAkRklMRSBdXTsgdGhlbgogIGRlYnVnICJGYWlsZWQgdG8gZmluZCByb290IGZpbGUgZm9yIHVuaXQgJ
FVOSVQgKCRGSUxFKSIKICBleGl0CmZpCmRlYnVnICJTZXJ2aWNlIGRlZmluaXRpb24gaXMgaW4gJEZJTEU
iCkVYRUNTVEFSVD0kKHNlZCAtbiAtZSAnL15FeGVjU3RhcnQ9LipcXCQvLC9bXlxcXSQvIHsgcy9eRXhlY
1N0YXJ0PS8vOyBwIH0nIC1lICcvXkV4ZWNTdGFydD0uKlteXFxdJC8geyBzL15FeGVjU3RhcnQ9Ly87IHA
gfScgJEZJTEUpCgppZiBbWyAkRU5WRklMRSBdXTsgdGhlbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU
1RBUlR9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkZJTEUKZWxzZQogIGVja G8
gJEVYRUNTVEFSVApmaQo=
        mode: 493
        path: /usr/local/bin/extractExecStart
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50LW5hbWVzc
GFjZS9tbnQgIiRAIgo=
        mode: 493
        path: /usr/local/bin/nsenterCmns
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Manages a mount namespace that both kubelet and crio can use
to share their container-specific mounts

          [Service]
          Type=oneshot
          RemainAfterExit=yes
          RuntimeDirectory=container-mount-namespace
          Environment=RUNTIME_DIRECTORY=%t/container-mount-namespace
          Environment=BIND_POINT=%t/container-mount-namespace/mnt
          ExecStartPre=bash -c "findmnt ${RUNTIME_DIRECTORY} || mount --make
-unbindable --bind ${RUNTIME_DIRECTORY} ${RUNTIME_DIRECTORY}"
          ExecStartPre=touch ${BIND_POINT}
          ExecStart=unshare --mount=${BIND_POINT} --propagation slave mount --make
-rshared /
          ExecStop=umount -R ${RUNTIME_DIRECTORY}
        enabled: true
        name: container-mount-namespace.service
```

```
      - dropins:
        - contents: |
            [Unit]
            Wants=container-mount-namespace.service
            After=container-mount-namespace.service

            [Service]
            ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env
ORIG_EXECSTART
            EnvironmentFile=-/%t/%N-execstart.env
            ExecStart=
            ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
                ${ORIG_EXECSTART}"
          name: 90-container-mount-namespace.conf
        name: crio.service
      - dropins:
        - contents: |
            [Unit]
            Wants=container-mount-namespace.service
            After=container-mount-namespace.service

            [Service]
            ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env
ORIG_EXECSTART
            EnvironmentFile=-/%t/%N-execstart.env
            ExecStart=
            ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
                ${ORIG_EXECSTART} --housekeeping-interval=30s"
          name: 90-container-mount-namespace.conf
        - contents: |
            [Service]
            Environment="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
            Environment="OPENSHIFT_EVICTION_MONITORING_PERIOD_DURATION=30s"
          name: 30-kubelet-interval-tuning.conf
        name: kubelet.service
```

Enabling Stream Control Transmission Protocol (SCTP)

SCTP is a key protocol used in RAN applications. This `MachineConfig` object adds the SCTP kernel module to the node to enable this protocol.

Procedure

- No configuration changes are needed. Use the provided settings:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: load-sctp-module
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
            mode: 420
            path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
            mode: 420
            path: /etc/modules-load.d/sctp-load.conf
```

Creating OperatorGroups for Operators

This configuration is provided to enable addition of the Operators needed to configure the platform post-installation. It adds the `Namespace` and `OperatorGroup` objects for the Local Storage Operator, Logging Operator, Performance Addon Operator, PTP Operator, and SRIOV Network Operator.

Procedure

- No configuration changes are needed. Use the provided settings:

Local Storage Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  name: openshift-local-storage
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-local-storage
  namespace: openshift-local-storage
spec:
  targetNamespaces:
    - openshift-local-storage
```

Logging Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  name: openshift-logging
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  targetNamespaces:
    - openshift-logging
```

Performance Addon Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-performance-addon-operator
spec: {}
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: performance-addon-operator
  namespace: openshift-performance-addon-operator
```

PTP Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-ptp
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
    - openshift-ptp
```

SRIOV Network Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
    name: openshift-sriov-network-operator
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
    - openshift-sriov-network-operator
```

Subscribing to the Operators

The subscription provides the location to download the Operators needed for platform configuration.

Procedure

- Use the following example to configure the subscription:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  channel: "stable" ①
  name: cluster-logging
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual ②
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: "stable" ③
  installPlanApproval: Automatic
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

```
  installPlanApproval: Manual
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: performance-addon-operator
  namespace: openshift-performance-addon-operator
spec:
  channel: "4.10" ④
  name: performance-addon-operator
  source: performance-addon-operator
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
    name: ptp-operator-subscription
    namespace: openshift-ptp
spec:
  channel: "stable" ⑤
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subscription
  namespace: openshift-sriov-network-operator
spec:
  channel: "stable" ⑥
  name: sriov-network-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
```

① Specify the channel to get the `cluster-logging` Operator.

② Specify `Manual` or `Automatic`. In `Automatic` mode, the Operator automatically updates to the latest versions in the channel as they become available in the registry. In `Manual` mode, new Operator versions are installed only after they are explicitly approved.

③ Specify the channel to get the `local-storage-operator` Operator.

④ Specify the channel to get the `performance-addon-operator` Operator.

⑤ Specify the channel to get the `ptp-operator` Operator.

⑥ Specify the channel to get the `sriov-network-operator` Operator.

Configuring logging locally and forwarding

To be able to debug a single node distributed unit (DU), logs need to be stored for further analysis.

Procedure

- Edit the `ClusterLogging` custom resource (CR) in the `openshift-logging` project:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging ①
 metadata:
  name: instance
  namespace: openshift-logging
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
  curation:
    type: "curator"
    curator:
      schedule: "30 3 * * *"
    managementState: Managed
---
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder ②
metadata:
  name: instance
  namespace: openshift-logging
spec:
  inputs:
    - infrastructure: {}
  outputs:
    - name: kafka-open
      type: kafka
      url: tcp://10.46.55.190:9092/test    ③
  pipelines:
    - inputRefs:
      - audit
      name: audit-logs
      outputRefs:
      - kafka-open
    - inputRefs:
      - infrastructure
      name: infrastructure-logs
      outputRefs:
      - kafka-open
```

① Updates the existing instance or creates the instance if it does not exist.

② Updates the existing instance or creates the instance if it does not exist.

③ Specifies the destination of the kafka server.

Configuring the Performance Addon Operator

This is a key configuration for the single node distributed unit (DU). Many of the real-time capabilities and service assurance are configured here.

Procedure

- Configure the performance addons using the following example:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: perfprofile-policy
spec:
  additionalKernelArgs:
    - idle=poll
    - rcupdate.rcu_normal_after_boot=0
  cpu:
    isolated: 2-19,22-39 ①
    reserved: 0-1,20-21 ②
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 32 ③
        size: 1G ④
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true ⑤
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true    ⑥
```

① Set the isolated CPUs. Ensure all of the HT pairs match.

② Set the reserved CPUs. In this case, a hyperthreaded pair is allocated on NUMA 0 and a pair on NUMA 1.

③ Set the huge page size.

④ Set the huge page number.

⑤ Set to `true` to isolate the CPUs from networking interrupts.

⑥ Set to `true` to install the real-time Linux kernel.

Configuring Precision Time Protocol (PTP)

In the far edge, the RAN uses PTP to synchronize the systems.

Procedure

- Configure PTP using the following example:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: du-ptp-slave
  namespace: openshift-ptp
spec:
  profile:
    - interface: ens5f0     ①
      name: slave
      phc2sysOpts: -a -r -n 24
      ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison ieee1588
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
        logSyncInterval -4
        logMinDelayReqInterval -4
        logMinPdelayReqInterval -4
        announceReceiptTimeout 3
        syncReceiptTimeout 0
        delayAsymmetry 0
        fault_reset_interval 4
        neighborPropDelayThresh 20000000
        masterOnly 0
```

```
        G.8275.portDS.localPriority 128
        #
        # Run time options
        #
        assume_two_step 0
        logging_level 6
        path_trace_enabled 0
        follow_up_info 0
        hybrid_e2e 0
        inhibit_multicast_service 0
        net_sync_monitor 0
        tc_spanning_tree 0
        tx_timestamp_timeout 50
        unicast_listen 0
        unicast_master_table 0
        unicast_req_duration 3600
        use_syslog 1
        verbose 0
        summary_interval 0
        kernel_leap 1
        check_fup_sync 0
        #
        # Servo Options
        #
        pi_proportional_const 0.0
        pi_integral_const 0.0
        pi_proportional_scale 0.0
        pi_proportional_exponent -0.3
        pi_proportional_norm_max 0.7
        pi_integral_scale 0.0
        pi_integral_exponent 0.4
        pi_integral_norm_max 0.3
        step_threshold 0.0
        first_step_threshold 0.00002
        max_frequency 900000000
        clock_servo pi
        sanity_freq_limit 200000000
        ntpshm_segment 0
        #
        # Transport options
        #
        transportSpecific 0x0
        ptp_dst_mac 01:1B:19:00:00:00
        p2p_dst_mac 01:80:C2:00:00:0E
        udp_ttl 1
        udp6_scope 0x0E
        uds_address /var/run/ptp4l
        #
        # Default interface options
        #
        clock_type OC
```

```
        network_transport UDPv4
        delay_mechanism E2E
        time_stamping hardware
        tsproc_mode filter
        delay_filter moving_median
        delay_filter_length 10
        egressLatency 0
        ingressLatency 0
        boundary_clock_jbod 0
        #
        # Clock description
        #
        productDescription ;;
        revisionData ;;
        manufacturerIdentity 00:00:00
        userDescription ;
        timeSource 0xA0
      ptp4lOpts: -2 -s --summary_interval -4
recommend:
  - match:
      - nodeLabel: node-role.kubernetes.io/master
    priority: 4
    profile: slave
```

① Sets the interface used for PTP.

Disabling Network Time Protocol (NTP)

After the system is configured for Precision Time Protocol (PTP), you need to remove NTP to prevent it from impacting the system clock.

Procedure

- No configuration changes are needed. Use the provided settings:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: disable-chronyd
spec:
  config:
    systemd:
      units:
        - contents: |
            [Unit]
            Description=NTP client/server
            Documentation=man:chronyd(8) man:chrony.conf(5)
            After=ntpdate.service sntp.service ntpd.service
            Conflicts=ntpd.service systemd-timesyncd.service
            ConditionCapability=CAP_SYS_TIME
            [Service]
            Type=forking
            PIDFile=/run/chrony/chronyd.pid
            EnvironmentFile=-/etc/sysconfig/chronyd
            ExecStart=/usr/sbin/chronyd $OPTIONS
            ExecStartPost=/usr/libexec/chrony-helper update-daemon
            PrivateTmp=yes
            ProtectHome=yes
            ProtectSystem=full
            [Install]
            WantedBy=multi-user.target
          enabled: false
          name: chronyd.service
    ignition:
      version: 2.2.0
```

Configuring single root I/O virtualization (SR-IOV)

SR-IOV is commonly used to enable the fronthaul and the midhaul networks.

Procedure

- Use the following configuration to configure SRIOV on a single node distributed unit (DU). Note that the first custom resource (CR) is required. The following CRs are examples.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
```

```
    node-role.kubernetes.io/master: ""
  disableDrain: true
  enableInjector: true
  enableOperatorWebhook: true
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nw-du-mh
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  resourceName: du_mh
  vlan: 150 ①
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-mh
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci ②
  isRdma: false
  nicSelector:
    pfNames:
      - ens7f0 ③
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 ④
  priority: 10
  resourceName: du_mh
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nw-du-fh
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  resourceName: du_fh
  vlan: 140 ⑤
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-fh
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice ⑥
  isRdma: true
  nicSelector:
```

```
    pfNames:
      - ens5f0 ⑦
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 ⑧
  priority: 10
  resourceName: du_fh
```

① Specifies the VLAN for the midhaul network.

② Select either `vfio-pci` or `netdevice`, as needed.

③ Specifies the interface connected to the midhaul network.

④ Specifies the number of VFs for the midhaul network.

⑤ The VLAN for the fronthaul network.

⑥ Select either `vfio-pci` or `netdevice`, as needed.

⑦ Specifies the interface connected to the fronthaul network.

⑧ Specifies the number of VFs for the fronthaul network.

Disabling the console Operator

The console-operator installs and maintains the web console on a cluster. When the node is centrally managed the Operator is not needed and makes space for application workloads.

Procedure

- You can disable the Operator using the following configuration file. No configuration changes are needed. Use the provided settings:

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "false"
    include.release.openshift.io/self-managed-high-availability: "false"
    include.release.openshift.io/single-node-developer: "false"
    release.openshift.io/create-only: "true"
  name: cluster
spec:
  logLevel: Normal
  managementState: Removed
  operatorLogLevel: Normal
```

Applying the distributed unit (DU) configuration to a single node cluster

Perform the following tasks to configure a single node cluster for a DU:

- Apply the required extra installation manifests at installation time.
- Apply the post-install configuration custom resources (CRs).

Applying the extra installation manifests

To apply the distributed unit (DU) configuration to the single node cluster, the following extra installation manifests need to be included during installation:

- Enable workload partitioning.
- Other `MachineConfig` objects – There is a set of `MachineConfig` custom resources (CRs) included by default. You can choose to include these additional `MachineConfig` CRs that are unique to their environment. It is recommended, but not required, to apply these CRs during installation in order to minimize the number of reboots that can occur during post-install configuration.

Applying the post-install configuration custom resources (CRs)

- After OpenShift Container Platform is installed on the cluster, use the following command to apply the CRs you configured for the distributed units (DUs):

```
$ oc apply -f <file_name>.yaml
```

Deploying distributed units at scale in a disconnected environment

Use zero touch provisioning (ZTP) to provision distributed units at new edge sites in a disconnected environment. The workflow starts when the site is connected to the network and ends with the CNF workload deployed and running on the site nodes.

Provisioning edge sites at scale

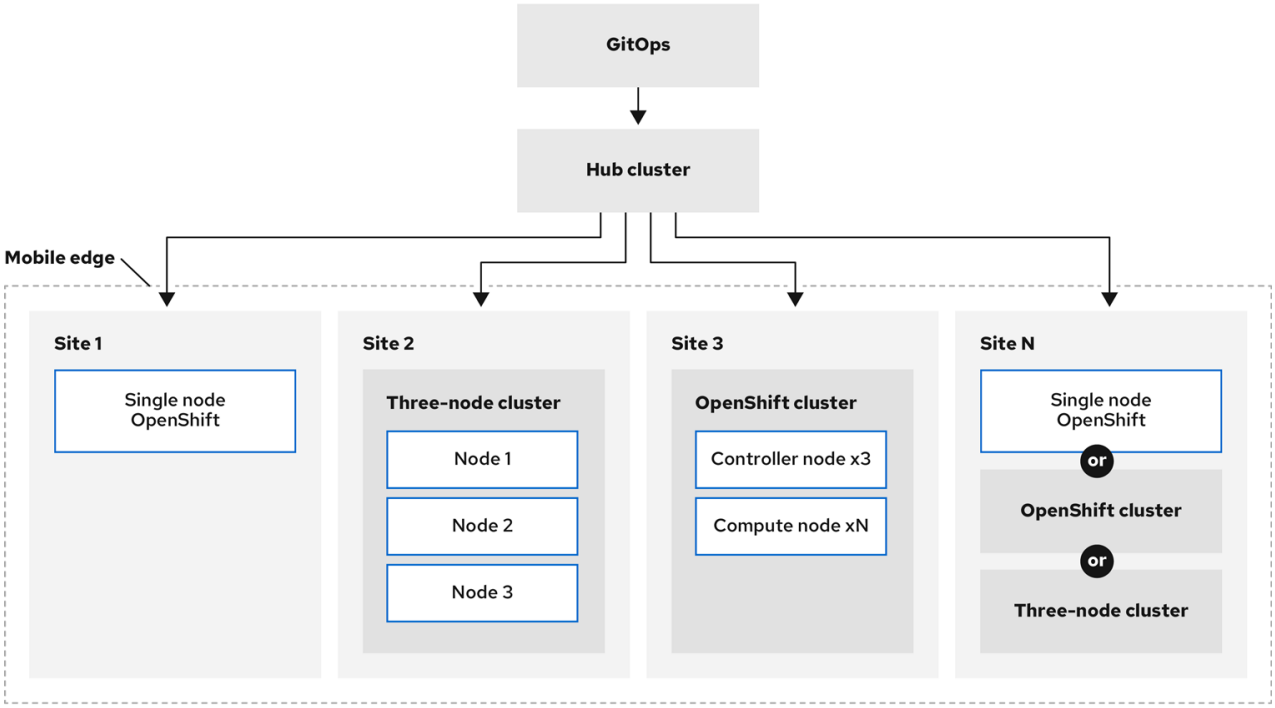
Telco edge computing presents extraordinary challenges with managing hundreds to tens of thousands of clusters in hundreds of thousands of locations. These challenges require fully-automated management solutions with, as closely as possible, zero human interaction.

Zero touch provisioning (ZTP) allows you to provision new edge sites with declarative configurations of bare-metal equipment at remote sites. Template or overlay configurations install OpenShift Container Platform features that are required for CNF workloads. End-to-end functional test suites are used to verify CNF related features. All configurations are declarative in nature.

You start the workflow by creating declarative configurations for ISO images that are delivered to the edge nodes to begin the installation process. The images are used to repeatedly provision large numbers of nodes efficiently and quickly, allowing you keep up with requirements from the field for far edge nodes.

Service providers are deploying a more distributed mobile network architecture allowed by the modular functional framework defined for 5G. This allows service providers to move from appliance-based radio access networks (RAN) to open cloud RAN architecture, gaining flexibility and agility in delivering services to end users.

The following diagram shows how ZTP works within a far edge framework.



About ZTP and distributed units on OpenShift clusters

You can install a distributed unit (DU) on OpenShift Container Platform clusters at scale with Red Hat Advanced Cluster Management (RHACM) using the assisted installer (AI) and the policy generator with core-reduction technology enabled. The DU installation is done using zero touch provisioning (ZTP) in a disconnected environment.

RHACM manages clusters in a hub-and-spoke architecture, where a single hub cluster manages many spoke clusters. RHACM applies radio access network (RAN) policies from predefined custom resources (CRs). Hub clusters running ACM provision and deploy the spoke clusters using ZTP and AI. DU installation follows the AI installation of OpenShift Container Platform on each cluster.

The AI service handles provisioning of OpenShift Container Platform on single node clusters, three-node clusters, or standard clusters running on bare metal. ACM ships with and deploys the AI when the `MultiClusterHub` custom resource is installed.

With ZTP and AI, you can provision OpenShift Container Platform clusters to run your DUs at scale. A high-level overview of ZTP for distributed units in a disconnected environment is as follows:

- A hub cluster running Red Hat Advanced Cluster Management (RHACM) manages a disconnected internal registry that mirrors the OpenShift Container Platform release images. The internal registry is used to provision the spoke clusters.
- You manage the bare metal host machines for your DUs in an inventory file that uses YAML for formatting. You store the inventory file in a Git repository.

- You install the DU bare metal host machines on site, and make the hosts ready for provisioning. To be ready for provisioning, the following is required for each bare metal host:
  - Network connectivity - including DNS for your network. Hosts should be reachable through the hub and managed spoke clusters. Ensure there is layer 3 connectivity between the hub and the host where you want to install your hub cluster.
  - Baseboard Management Controller (BMC) details for each host - ZTP uses BMC details to connect the URL and credentials for accessing the BMC. ZTP manages the spoke cluster definition CRs, with the exception of the `BMCSecret` CR, which you create manually. These define the relevant elements for the managed clusters.

The GitOps approach

ZTP uses the GitOps deployment set of practices for infrastructure deployment that allows developers to perform tasks that would otherwise fall under the purview of IT operations. GitOps achieves these tasks using declarative specifications stored in Git repositories, such as YAML files and other defined patterns, that provide a framework for deploying the infrastructure. The declarative output is leveraged by the Open Cluster Manager (OCM) for multisite deployment.

One of the motivators for a GitOps approach is the requirement for reliability at scale. This is a significant challenge that GitOps helps solve.

GitOps addresses the reliability issue by providing traceability, RBAC, and a single source of truth for the desired state of each site. Scale issues are addressed by GitOps providing structure, tooling, and event driven operations through webhooks.

Zero touch provisioning building blocks

Red Hat Advanced Cluster Management (RHACM) leverages zero touch provisioning (ZTP) to deploy single-node OpenShift Container Platform clusters, three-node clusters, and standard clusters. The initial site plan is divided into smaller components and initial configuration data is stored in a Git repository. ZTP uses a declarative GitOps approach to deploy these clusters.

The deployment of the clusters includes:

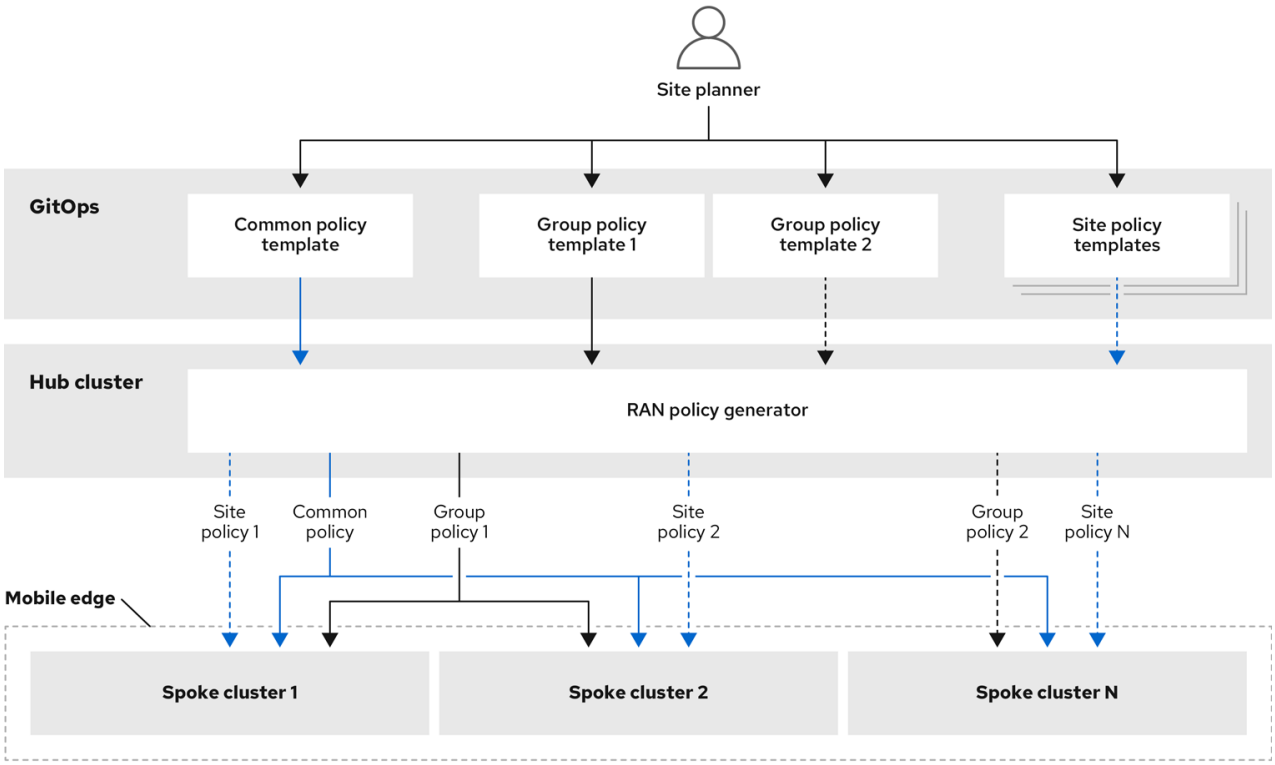
- Installing the host operating system (RHCOS) on a blank server.
- Deploying OpenShift Container Platform.
- Creating cluster policies and site subscriptions.
- Leveraging a GitOps deployment topology for a develop once, deploy anywhere model.
- Making the necessary network configurations to the server operating system.
- Deploying profile Operators and performing any needed software-related configuration, such as performance profile, PTP, and SR-IOV.
- Downloading images needed to run workloads (CNFs).

How to plan your RAN policies

Zero touch provisioning (ZTP) uses Red Hat Advanced Cluster Management (RHACM) to apply the radio access network (RAN) configuration using a policy-based governance approach to apply the configuration.

The policy generator or `PolicyGen` is a part of the GitOps ZTP tooling that facilitates creating RHACM policies from a set of predefined custom resources. There are three main items: policy categorization, source CR policy, and the `PolicyGenTemplate` CR. `PolicyGen` uses these to generate the policies and their placement bindings and rules.

The following diagram shows how the RAN policy generator interacts with GitOps and RHACM.



217_OpenShift_0222

RAN policies are categorized into three main groups:

Common

A policy that exists in the `Common` category is applied to all clusters to be represented by the site plan. Cluster types include single node, three-node, and standard clusters.

Groups

A policy that exists in the `Groups` category is applied to a group of clusters. Every group of clusters could have their own policies that exist under the `Groups` category. For example, `Groups/group1` can have its own policies that are applied to the clusters belonging to `group1`. You can also define a group for each cluster type: single node, three-node, and standard clusters.

Sites

A policy that exists in the `Sites` category is applied to a specific cluster. Any cluster could have its own policies that exist in the `Sites` category. For example, `Sites/cluster1` has its own policies applied to `cluster1`. You can also define an example site-specific configuration for each cluster type: single node, three-node, and standard clusters.

Low latency for distributed units (DUs)

Low latency is an integral part of the development of 5G networks. Telecommunications networks require as little signal delay as possible to ensure quality of service in a variety of critical use cases.

Low latency processing is essential for any communication with timing constraints that affect functionality and security. For example, 5G Telco applications require a guaranteed one millisecond one-way latency to meet Internet of Things (IoT) requirements. Low latency is also critical for the future development of autonomous vehicles, smart factories, and online gaming. Networks in these environments require almost a real-time flow of data.

Low latency systems are about guarantees with regards to response and processing times. This includes keeping a communication protocol running smoothly, ensuring device security with fast responses to error conditions, or just making sure a system is not lagging behind when receiving a lot of data. Low latency is key for optimal synchronization of radio transmissions.

OpenShift Container Platform enables low latency processing for DUs running on COTS hardware by using a number of technologies and specialized hardware devices:

Real-time kernel for RHCOS

Ensures workloads are handled with a high degree of process determinism.

CPU isolation

Avoids CPU scheduling delays and ensures CPU capacity is available consistently.

NUMA awareness

Aligns memory and huge pages with CPU and PCI devices to pin guaranteed container memory and huge pages to the NUMA node. This decreases latency and improves performance of the node.

Huge pages memory management

Using huge page sizes improves system performance by reducing the amount of system resources required to access page tables.

Precision timing synchronization using PTP

Allows synchronization between nodes in the network with sub-microsecond accuracy.

Preparing the disconnected environment

Before you can provision distributed units (DU) at scale, you must install Red Hat Advanced Cluster Management (RHACM), which handles the provisioning of the DUs.

RHACM is deployed as an Operator on the OpenShift Container Platform hub cluster. It controls clusters and applications from a single console with built-in security policies. RHACM provisions and manage your DU hosts. To install RHACM in a disconnected environment, you create a mirror registry that mirrors the Operator Lifecycle Manager (OLM) catalog that contains the required Operator images. OLM manages, installs, and upgrades Operators and their dependencies in the cluster.

You also use a disconnected mirror host to serve the RHCOS ISO and RootFS disk images that provision the DU bare-metal host operating system.

Additional resources

- For more information about creating the disconnected mirror registry, see Creating a mirror registry.
- For more information about mirroring OpenShift Platform image to the disconnected registry, see Mirroring images for a disconnected installation.

Adding RHCOS ISO and RootFS images to the disconnected mirror host

Before you install a cluster on infrastructure that you provision, you must create Red Hat Enterprise Linux CoreOS (RHCOS) machines for it to use. Use a disconnected mirror to host the RHCOS images you require to provision your distributed unit (DU) bare-metal hosts.

Prerequisites

- Deploy and configure an HTTP server to host the RHCOS image resources on the network. You must be able to access the HTTP server from your computer, and from the machines that you create.

The RHCOS images might not change with every release of OpenShift Container Platform. You must download images with the highest version that is less than or equal to the OpenShift Container Platform version that you install. Use the image versions that match your OpenShift Container Platform version if they are available. You require ISO and RootFS images to install RHCOS on the DU hosts. RHCOS qcow2 images are not supported for this installation type.

Procedure

1. Log in to the mirror host.
2. Obtain the RHCOS ISO and RootFS images from mirror.openshift.com, for example:
   a. Export the required image names and OpenShift Container Platform version as environment variables:

```
$ export ISO_IMAGE_NAME=<iso_image_name> ①
```

```
$ export ROOTFS_IMAGE_NAME=<rootfs_image_name> ②
```

```
$ export OCP_VERSION=<ocp_version> ③
```

① ISO image name, for example, `rhcos-4.10.0-fc.1-x86_64-live.x86_64.iso`

② RootFS image name, for example, `rhcos-4.10.0-fc.1-x86_64-live-rootfs.x86_64.img`

③ OpenShift Container Platform version, for example, `latest-4.10`

b. Download the required images:

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/pre-release/${OCP_VERSION}/${ISO_IMAGE_NAME} -O
/var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/pre-release/${OCP_VERSION}/${ROOTFS_IMAGE_NAME} -O
/var/www/html/${ROOTFS_IMAGE_NAME}
```

Verification steps

- Verify that the images downloaded successfully and are being served on the disconnected mirror host, for example:

```
$ wget http://$(hostname)/${ISO_IMAGE_NAME}
```

Expected output

```
...
Saving to: rhcos-4.10.0-fc.1-x86_64-live.x86_64.iso
rhcos-4.10.0-fc.1-x86_64-  11%[====>    ]  10.01M  4.71MB/s
...
```

Installing Red Hat Advanced Cluster Management in a disconnected environment

You use Red Hat Advanced Cluster Management (RHACM) on a hub cluster in the disconnected environment to manage the deployment of distributed unit (DU) profiles on multiple managed spoke clusters.

Prerequisites

- Install the OpenShift Container Platform CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.
- Configure a disconnected mirror registry for use in the cluster.

If you want to deploy Operators to the spoke clusters, you must also add them to this registry. See Mirroring an Operator catalog for more information.

Procedure

- Install RHACM on the hub cluster in the disconnected environment. See Installing RHACM in a disconnected environment.

Enabling assisted installer service on bare metal

The Assisted Installer Service (AIS) deploys OpenShift Container Platform clusters. Red Hat Advanced Cluster Management (RHACM) ships with AIS. AIS is deployed when you enable the MultiClusterHub Operator on the RHACM hub cluster.

For distributed units (DUs), RHACM supports OpenShift Container Platform deployments that run on a single bare-metal host, three-node clusters, or standard clusters. In the case of single node clusters or three-node clusters, all nodes act as both control plane and worker nodes.

Prerequisites

- Install OpenShift Container Platform 4.10 on a hub cluster.
- Install RHACM and create the `MultiClusterHub` resource.
- Create persistent volume custom resources (CR) for database and file system storage.
- You have installed the OpenShift CLI (`oc`).

Procedure

1. Modify the `HiveConfig` resource to enable the feature gate for Assisted Installer:

```
 $ oc patch hiveconfig hive --type merge -p
'{"spec":{"targetNamespace":"hive","logLevel":"debug","featureGates":{"custom":
{"enabled":["AlphaAgentInstallStrategy"]},"featureSet":"Custom"}}}'
```

2. Modify the `Provisioning` resource to allow the Bare Metal Operator to watch all namespaces:

```
 $ oc patch provisioning provisioning-configuration --type merge -p
'{"spec":{"watchAllNamespaces": true }}'
```

3. Create the `AgentServiceConfig` CR.
   a. Save the following YAML in the `agent_service_config.yaml` file:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
 name: agent
spec:
  databaseStorage:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <db_volume_size> ①
  filesystemStorage:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <fs_volume_size> ②
  osImages: ③
    - openshiftVersion: "<ocp_version>" ④
      version: "<ocp_release_version>" ⑤
      url: "<iso_url>" ⑥
      rootFSUrl: "<root_fs_url>" ⑦
      cpuArchitecture: "x86_64"
```

① Volume size for the `databaseStorage` field, for example `10Gi`.

② Volume size for the `filesystemStorage` field, for example `20Gi`.

③ List of OS image details. Example describes a single OpenShift Container Platform OS version.

④ OpenShift Container Platform version to install, for example, `4.8`.

⑤ Specific install version, for example, `47.83.202103251640-0`.

⑥ ISO url, for example, `https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.7/4.7.7/rhcos-4.7.7-x86_64-live.x86_64.iso`.

⑦ Root FS image URL, for example, `https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.7/4.7.7/rhcos-live-rootfs.x86_64.img`

b. Create the `AgentServiceConfig` CR by running the following command:

```
$ oc create -f agent_service_config.yaml
```

Example output

```
agentserviceconfig.agent-install.openshift.io/agent created
```

ZTP custom resources

Zero touch provisioning (ZTP) uses custom resource (CR) objects to extend the Kubernetes API or introduce your own API into a project or a cluster. These CRs contain the site-specific data required to install and configure a cluster for RAN applications.

A custom resource definition (CRD) file defines your own object kinds. Deploying a CRD into the managed cluster causes the Kubernetes API server to begin serving the specified CR for the entire lifecycle.

For each CR in the `<site>.yaml` file on the managed cluster, ZTP uses the data to create installation CRs in a directory named for the cluster.

ZTP provides two ways for defining and installing CRs on managed clusters: a manual approach when you are provisioning a single cluster and an automated approach when provisioning multiple clusters.

Manual CR creation for single clusters

Use this method when you are creating CRs for a single cluster. This is a good way to test your CRs before deploying on a larger scale.

Automated CR creation for multiple managed clusters

Use the automated SiteConfig method when you are installing multiple managed clusters, for example, in batches of up to 100 clusters. SiteConfig uses ArgoCD as the engine for the GitOps method of site deployment. After completing a site plan that contains all of the required parameters for deployment, a policy generator creates the manifests and applies them to the hub cluster.

Both methods create the CRs shown in the following table. On the cluster site, an automated Discovery image ISO file creates a directory with the site name and a file with the cluster name. Every cluster has its own namespace, and all of the CRs are under that namespace. The namespace and the CR names match the cluster name.

Resource	Description	Usage
`BareMetalHost`	Contains the connection information for the Baseboard Management Controller (BMC) of the target bare-metal host.	Provides access to the BMC in order to load and boot the Discovery image ISO on the target server by using the Redfish protocol.
`InfraEnv`	Contains information for pulling OpenShift Container Platform onto the target bare-metal host.	Used with ClusterDeployment to generate the Discovery ISO for the managed cluster.

Resource	Description	Usage
AgentClusterInstall	Specifies the managed cluster's configuration such as networking and the number of supervisor (control plane) nodes. Shows the kubeconfig and credentials when the installation is complete.	Specifies the managed cluster configuration information and provides status during the installation of the cluster.
ClusterDeployment	References the AgentClusterInstall to use.	Used with InfraEnv to generate the Discovery ISO for the managed cluster.
NMStateConfig	Provides network configuration information such as MAC to IP mapping, DNS server, default route, and other network settings. This is not needed if DHCP is used.	Sets up a static IP address for the managed cluster's Kube API server.
Agent	Contains hardware information about the target bare-metal host.	Created automatically on the hub when the target machine's Discovery image ISO boots.
ManagedCluster	When a cluster is managed by the hub, it must be imported and known. This Kubernetes object provides that interface.	The hub uses this resource to manage and show the status of managed clusters.
KlusterletAddonConfig	Contains the list of services provided by the hub to be deployed to a ManagedCluster.	Tells the hub which addon services to deploy to a ManagedCluster.
Namespace	Logical space for ManagedCluster resources existing on the hub. Unique per site.	Propagates resources to the ManagedCluster.
Secret	Two custom resources are created: BMC Secret and Image Pull Secret.	• BMC Secret authenticates into the target bare-metal host using its username and password. • Image Pull Secret contains authentication information for the OpenShift Container Platform image installed on the target bare-metal host.

Resource	Description	Usage
`ClusterImageSet`	Contains OpenShift Container Platform image information such as the repository and image name.	Passed into resources to provide OpenShift Container Platform images.

ZTP support for single node clusters, three-node clusters, and standard clusters requires updates to these CRs, including multiple instantiations of some.

ZTP provides support for deploying single node clusters, three-node clusters, and standard OpenShift clusters. This includes the installation of OpenShift and deployment of the distributed units (DUs) at scale.

The overall flow is identical to the ZTP support for single node clusters, with some differences in configuration depending on the type of cluster:

`SiteConfig` file:

- For single node clusters, the `SiteConfig` file must have exactly one entry in the `nodes` section.
- For three-node clusters, the `SiteConfig` file must have exactly three entries defined in the `nodes` section.
- For standard clusters, the `SiteConfig` file must have exactly three entries in the `nodes` section with `role: master` and one or more additional entries with `role: worker`.

`PolicyGenTemplate` file:

- The example common `PolicyGenTemplate` file is common across all types of clusters.
- There are example group `PolicyGenTemplate` files for single node, three-node, and standard clusters.
- Site-specific `PolicyGenTemplate` files are still specific to each site.

PolicyGenTemplate CRs for RAN deployments

You use `PolicyGenTemplate` custom resources (CRs) to customize the configuration applied to the cluster using the GitOps zero touoch provisioning (ZTP) pipeline. The baseline configuration, obtained from the GitOps ZTP container, is designed to provide a set of critical features and node tuning settings that ensure the cluster can support the stringent performance and resource utilization constraints typical of RAN Distributed Unit (DU) applications. Changes or omissions from the baseline configuration can affect feature availability, performance, and resource utilization. Use `PolicyGenTemplate` CRs as the basis to create a hierarchy of configuration files tailored to your specific site requirements.

The baseline `PolicyGenTemplate` CRs that are defined for RAN DU cluster configuration can be extracted from the GitOps ZTP `ztp-site-generator`. See "Preparing the ZTP Git repository" for further details.

The `PolicyGenTemplate` CRs can be found in the `./out/argocd/example/policygentemplates` folder.

The reference architecture has common, group, and site-specific configuration CRs. Each `PolicyGenTemplate` CR refers to other CRs that can be found in the `./out/source-crs` folder.

The `PolicyGenTemplate` CRs relevant to RAN cluster configuration are described below. Variants are provided for the group `PolicyGenTemplate` CRs to account for differences in single-node, three-node compact, and standard cluster configurations. Similarly, site-specific configuration variants are provided for single-node clusters and multi-node (compact or standard) clusters. Use the group and site-specific configuration variants that are relevant for your deployment.

Table 6. PolicyGenTemplate CRs for RAN deployments

PolicyGenTemplate CR	Description
`common-ranGen.yaml`	Contains a set of common RAN CRs that get applied to all clusters. These CRs subscribe to a set of operators providing cluster features typical for RAN as well as baseline cluster tuning.
`group-du-3node-ranGen.yaml`	Contains the RAN policies for three-node clusters only.
`group-du-sno-ranGen.yaml`	Contains the RAN policies for single-node clusters only.
`group-du-standard-ranGen.yaml`	Contains the RAN policies for standard three control-plane clusters.

Additional resources

- For more information about extracting the `/argocd` directory from the `ztp-site-generator` container image, see Preparing the ZTP Git repository.

About the PolicyGenTemplate

The `PolicyGenTemplate.yaml` file is a custom resource definition (CRD) that tells the `PolicyGen` policy generator what CRs to include in the configuration, how to categorize the CRs into the generated policies, and what items in those CRs need to be updated with overlay content.

The following example shows a `PolicyGenTemplate.yaml` file:

```
---
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno"
  namespace: "group-du-sno-policies"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  sourceFiles:
```

```
    - fileName: ConsoleOperatorDisable.yaml
      policyName: "console-policy"
    - fileName: ClusterLogForwarder.yaml
      policyName: "log-forwarder-policy"
      spec:
        outputs:
          - type: "kafka"
            name: kafka-open
            # below url is an example
            url: tcp://10.46.55.190:9092/test
        pipelines:
          - name: audit-logs
            inputRefs:
             - audit
            outputRefs:
             - kafka-open
          - name: infrastructure-logs
            inputRefs:
             - infrastructure
            outputRefs:
             - kafka-open
    - fileName: ClusterLogging.yaml
      policyName: "log-policy"
      spec:
        curation:
          curator:
            schedule: "30 3 * * *"
        collection:
          logs:
            type: "fluentd"
            fluentd: {}
    - fileName: MachineConfigSctp.yaml
      policyName: "mc-sctp-policy"
      metadata:
        labels:
          machineconfiguration.openshift.io/role: master
    - fileName: PtpConfigSlave.yaml
      policyName: "ptp-config-policy"
      metadata:
        name: "du-ptp-slave"
      spec:
        profile:
        - name: "slave"
          interface: "ens5f0"
          ptp4lOpts: "-2 -s --summary_interval -4"
          phc2sysOpts: "-a -r -n 24"
    - fileName: SriovOperatorConfig.yaml
      policyName: "sriov-operconfig-policy"
      spec:
        disableDrain: true
    - fileName: MachineConfigAcceleratedStartup.yaml
```

```
      policyName: "mc-accelerated-policy"
      metadata:
        name: 04-accelerated-container-startup-master
        labels:
          machineconfiguration.openshift.io/role: master
    - fileName: DisableSnoNetworkDiag.yaml
      policyName: "disable-network-diag"
      metadata:
        labels:
          machineconfiguration.openshift.io/role: master
```

The `group-du-ranGen.yaml` file defines a group of policies under a group named `group-du`. A Red Hat Advanced Cluster Management (RHACM) policy is generated for every source file that exists in `sourceFiles`. And, a single placement binding and placement rule is generated to apply the cluster selection rule for `group-du` policies.

Using the source file `PtpConfigSlave.yaml` as an example, the `PtpConfigSlave` has a definition of a `PtpConfig` custom resource (CR). The generated policy for the `PtpConfigSlave` example is named `group-du-ptp-config-policy`. The `PtpConfig` CR defined in the generated `group-du-ptp-config-policy` is named `du-ptp-slave`. The `spec` defined in `PtpConfigSlave.yaml` is placed under `du-ptp-slave` along with the other `spec` items defined under the source file.

The following example shows the `group-du-ptp-config-policy`:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: group-du-ptp-config-policy
  namespace: groups-sub
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
spec:
    remediationAction: enforce
    disabled: false
    policy-templates:
        - objectDefinition:
            apiVersion: policy.open-cluster-management.io/v1
            kind: ConfigurationPolicy
            metadata:
                name: group-du-ptp-config-policy-config
            spec:
                remediationAction: enforce
                severity: low
                namespaceselector:
                    exclude:
                        - kube-*
                    include:
                        - '*'
```

```
            object-templates:
                - complianceType: musthave
                  objectDefinition:
                    apiVersion: ptp.openshift.io/v1
                    kind: PtpConfig
                    metadata:
                        name: slave
                        namespace: openshift-ptp
                    spec:
                        recommend:
                            - match:
                            - nodeLabel: node-role.kubernetes.io/worker-du
                              priority: 4
                              profile: slave
                        profile:
                            - interface: ens5f0
                              name: slave
                              phc2sysOpts: -a -r -n 24
                              ptp4lConf: |
                                [global]
                                #
                                # Default Data Set
                                #
                                twoStepFlag 1
                                slaveOnly 0
                                priority1 128
                                priority2 128
                                domainNumber 24
                                .....
```

Best practices when customizing PolicyGenTemplate CRs

Consider the following best practices when customizing site configuration `PolicyGenTemplate` CRs:

- Use as few policies as necessary. Using fewer policies means using less resources. Each additional policy creates overhead for the hub cluster and the deployed spoke cluster. CRs are combined into policies based on the `policyName` field in the `PolicyGenTemplate` CR. CRs in the same `PolicyGenTemplate` which have the same value for `policyName` are managed under a single policy.
- Use a single catalog source for all Operators. In disconnected environments, configure the registry as a single index containing all Operators. Each additional `CatalogSource` on the spoke clusters increases CPU usage.
- `MachineConfig` CRs should be included as `extraManifests` in the `SiteConfig` CR so that they are applied during installation. This can reduce the overall time taken until the cluster is ready to deploy applications.

- `PolicyGenTemplates` should override the channel field to explicitly identify the desired version. This ensures that changes in the source CR during upgrades does not update the generated subscription.

Additional resources

- For details about best practice for scaling clusters with Red Hat Advanced Cluster Management (RHACM), see ACM performance and scalability considerations.

Scaling the hub cluster to managing large numbers of spoke clusters is affected by the number of policies created on the hub cluster. Grouping multiple configuration CRs into a single or limited number of policies is one way to reduce the overall number of policies on the hub cluster. When using the common/group/site hierarchy of policies for managing site configuration, it is especially important to combine site-specific configuration into a single policy.

Creating the PolicyGenTemplate CR

Use this procedure to create the `PolicyGenTemplate` custom resource (CR) for your site in your local clone of the Git repository.

Procedure

1. Choose an appropriate example from `out/argocd/example/policygentemplates`. This directory demonstrates a three-level policy framework that represents a well-supported low-latency profile tuned for the needs of 5G Telco DU deployments:
   - A single `common-ranGen.yaml` file that should apply to all types of sites.
   - A set of shared `group-du-*-ranGen.yaml` files, each of which should be common across a set of similar clusters.
   - An example `example-*-site.yaml` that can be copied and updated for each individual site.
2. Ensure that the labels defined in your `PolicyGenTemplate` `bindingRules` section correspond to the labels that are defined in the `SiteConfig` files of the clusters you are managing.
3. Ensure that the content of the overlaid spec files matches your desired end state. As a reference, the `out/source-crs` directory contains the full list of `source-crs` available to be included and overlaid by your `PolicyGenTemplate` templates.

Depending on the specific requirements of your clusters, you might need more than a single group policy per cluster type, especially considering that the example group policies each have a single `PerformancePolicy.yaml` file that can only be shared across a set of clusters if those clusters consist of identical hardware configurations.

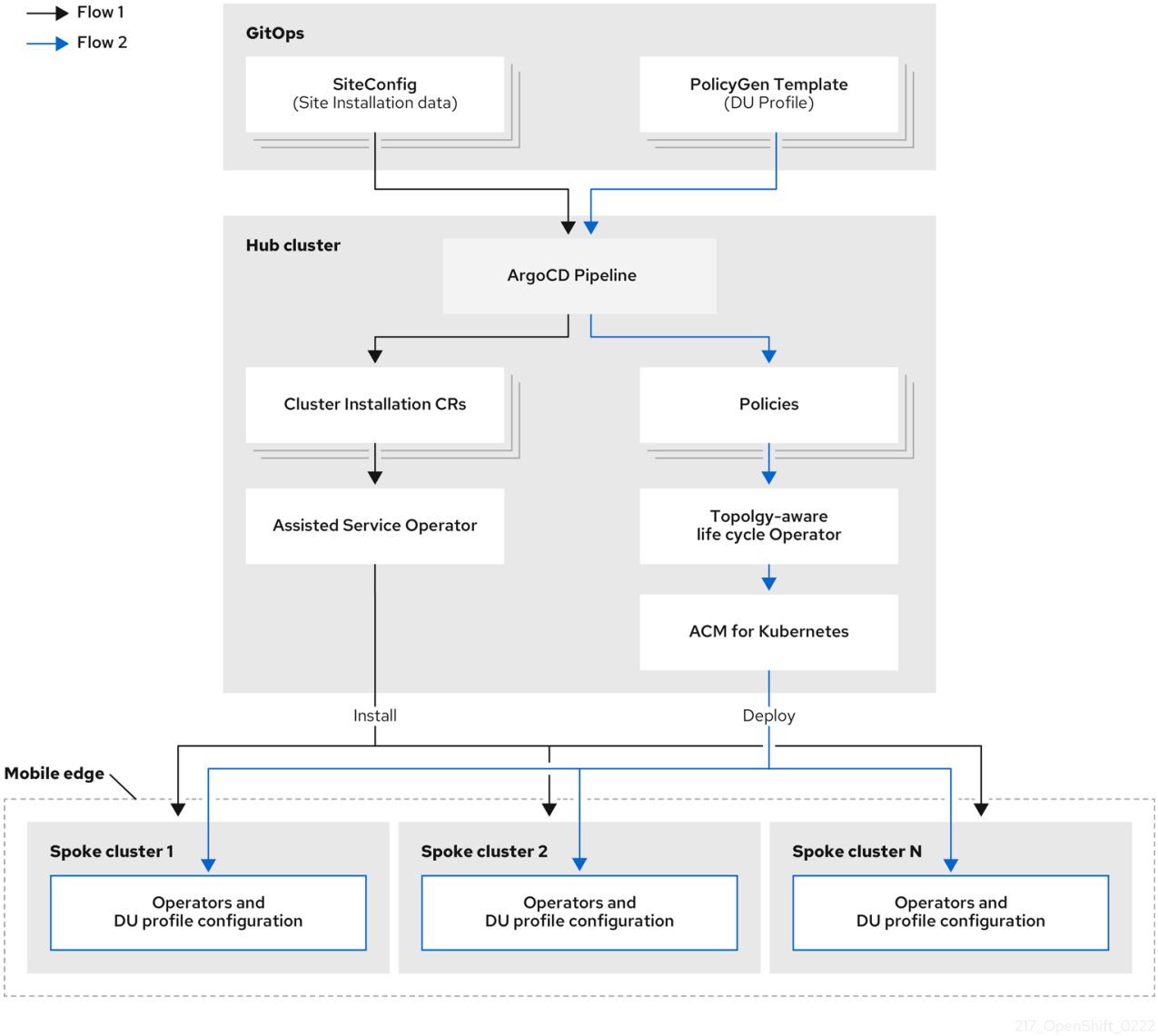
4. Define all the policy namespaces in a YAML file similar to the example `out/argocd/example/policygentemplates/ns.yaml` file.
5. Add all the `PolicyGenTemplate` files and `ns.yaml` file to the `kustomization.yaml` file, similar to the example `out/argocd/example/policygentemplates/kustomization.yaml` file.

6. Commit the `PolicyGenTemplate` CRs, `ns.yaml` file, and the associated `kustomization.yaml` file in the Git repository.

Creating ZTP custom resources for multiple managed clusters

If you are installing multiple managed clusters, zero touch provisioning (ZTP) uses ArgoCD and `SiteConfig` files to manage the processes that create the CRs and generate and apply the policies for multiple clusters, in batches of no more than 100, using the GitOps approach.

Installing and deploying the clusters is a two stage process, as shown here:



Using PolicyGenTemplate CRs to override source CRs content

`PolicyGenTemplate` CRs allow you to overlay additional configuration details on top of the base source CRs provided in the `ztp-site-generate` container. You can think of `PolicyGenTemplate` CRs as a logical merge or patch to the base CR. Use `PolicyGenTemplate` CRs to update a single field of the base CR, or overlay the entire contents of the base CR. You can update values and insert

fields that are not in the base CR.

The following example procedure describes how to update fields in the generated `PerformanceProfile` CR for the reference configuration based on the `PolicyGenTemplate` CR in the `group-du-sno-ranGen.yaml` file. Use the procedure as a basis for modifying other parts of the `PolicyGenTemplate` based on your requirements.

Prerequisites

- Create a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and be defined as a source repository for Argo CD.

Procedure

1. Review the baseline source CR for existing content. You can review the source CRs listed in the reference `PolicyGenTemplate` CRs by extracting them from the zero touch provisioning (ZTP) container.

   a. Create an `/out` folder:

   ```
   $ mkdir -p ./out
   ```

   b. Extract the source CRs:

   ```
   $ podman run --log-driver=none --rm quay.io/openshift-kni/ztp-site-
   generator:4.10.0 extract /home/ztp --tar | tar x -C ./out
   ```

2. Review the baseline `PerformanceProfile` CR in `./out/source-crs/PerformanceProfile.yaml`:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
  - "idle=poll"
  - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
```

Any fields in the source CR which contain `$…` are removed from the generated CR if they are not provided in the `PolicyGenTemplate` CR.

3. Update the `PolicyGenTemplate` entry for `PerformanceProfile` in the `group-du-sno-ranGen.yaml` reference file. The following example `PolicyGenTemplate` CR stanza supplies appropriate CPU specifications, sets the `hugepages` configuration, and adds a new field that sets `globallyDisableIrqLoadBalancing` to false.

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    name: openshift-node-performance-profile
  spec:
    cpu:
      # These must be tailored for the specific hardware platform
      isolated: "2-19,22-39"
      reserved: "0-1,20-21"
    hugepages:
      defaultHugepagesSize: 1G
      pages:
        - size: 1G
          count: 10
    globallyDisableIrqLoadBalancing: false
```

4. Commit the `PolicyGenTemplate` change in Git, and then push to the Git repository being monitored by the GitOps ZTP argo CD application.

Example output

The ZTP application generates an ACM policy that contains the generated `PerformanceProfile` CR. The contents of that CR are derived by merging the `metadata` and `spec` contents from the `PerformanceProfile` entry in the `PolicyGenTemplate` onto the source CR. The resulting CR has the following content:

```
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: openshift-node-performance-profile
spec:
    additionalKernelArgs:
        - idle=poll
        - rcupdate.rcu_normal_after_boot=0
    cpu:
        isolated: 2-19,22-39
        reserved: 0-1,20-21
    globallyDisableIrqLoadBalancing: false
    hugepages:
        defaultHugepagesSize: 1G
        pages:
            - count: 10
              size: 1G
    machineConfigPoolSelector:
        pools.operator.machineconfiguration.openshift.io/master: ""
    net:
        userLevelNetworking: true
    nodeSelector:
        node-role.kubernetes.io/master: ""
    numa:
        topologyPolicy: restricted
    realTimeKernel:
        enabled: true
```

In the `/source-crs` folder that you extract from the `ztp-site-generate` container, the `$` syntax is not used for template substitution as implied by the syntax. Rather, if the `policyGen` tool sees the `$` prefix for a string and you do not specify a value for that field in the related `PolicyGenTemplate` CR, the field is omitted from the output CR entirely.

An exception to this is the `$mcp` variable in `/source-crs` YAML files that is substituted with the specified value for `mcp` from the `PolicyGenTemplate` CR. For example, in `example/policygentemplates/group-du-standard-ranGen.yaml`, the value for `mcp` is `worker`:

```
spec:
  bindingRules:
    group-du-standard: ""
  mcp: "worker"
```

The `policyGen` tool replace instances of `$mcp` with `worker` in the output CRs.

Configuring PTP fast events using PolicyGenTemplate CRs

You can configure PTP fast events for vRAN clusters that are deployed using the GitOps Zero Touch Provisioning (ZTP) pipeline. Use `PolicyGenTemplate` custom resources (CRs) as the basis to create a hierarchy of configuration files tailored to your specific site requirements.

Prerequisites

- Create a Git repository where you manage your custom site configuration data.

Procedure

1. Add the following YAML into `.spec.sourceFiles` in the `common-ranGen.yaml` file to configure the AMQP Operator:

```
#AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
```

2. Apply the following `PolicyGenTemplate` changes to `group-du-3node-ranGen.yaml`, `group-du-sno-ranGen.yaml`, or `group-du-standard-ranGen.yaml` files according to your requirements:

   a. In `.sourceFiles`, add the `PtpOperatorConfig` CR file that configures the AMQ transport host to the `config-policy`:

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
```

   b. Configure the `linuxptp` and `phc2sys` for the PTP clock type and interface. For example, add the following stanza into `.sourceFiles`:

```
- fileName: PtpConfigSlave.yaml ①
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
    - name: "slave"
      interface: "ens5f1" ②
      ptp4lOpts: "-2 -s --summary_interval -4" ③
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16"
    ptpClockThreshold: ④
      holdOverTimeout: 30 #secs
      maxOffsetThreshold: 100  #nano secs
      minOffsetThreshold: -100 #nano secs
```

① Can be one `PtpConfigMaster.yaml`, `PtpConfigSlave.yaml`, or `PtpConfigSlaveCvl.yaml` depending on your requirements. `PtpConfigSlaveCvl.yaml` configures `linuxptp` services for an Intel E810 Columbiaville NIC. For configurations based on `group-du-sno-ranGen.yaml` or `group-du-3node-ranGen.yaml`, use `PtpConfigSlave.yaml`.

② Device specific interface name.

③ You must append the `--summary_interval -4` value to `ptp4lOpts` in `.spec.sourceFiles.spec.profile` to enable PTP fast events.

④ `ptpClockThreshold` configues how long the clock stays in clock holdover state. Holdover state is the period between local and master clock synchronizations. Offset is the time difference between the local and master clock.

3. Apply the following `PolicyGenTemplate` changes to your specific site YAML files, for example, `example-sno-site.yaml`:

   a. In `.sourceFiles`, add the `Interconnect` CR file that configures the AMQ router to the `config-policy`:

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

4. Merge any other required changes and files with your custom site repository.
5. Push the changes to your site configuration repository to deploy PTP fast events to new sites using GitOps ZTP.

Configuring UEFI secure boot for clusters using PolicyGenTemplate CRs

You can configure UEFI secure boot for vRAN clusters that are deployed using the GitOps zero touch provisioning (ZTP) pipeline.

Prerequisites

- Create a Git repository where you manage your custom site configuration data.

Procedure

1. Create the following `MachineConfig` resource and save it in the `uefi-secure-boot.yaml` file:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: uefi-secure-boot
spec:
  config:
    ignition:
      version: 3.1.0
  kernelArguments:
    - efi=runtime
```

2. In your Git repository custom `/siteconfig` directory, create a `/sno-extra-manifest` folder and add the `uefi-secure-boot.yaml` file, for example:

```
siteconfig
├──── site1-sno-du.yaml
├──── site2-standard-du.yaml
└──── sno-extra-manifest
    └──── uefi-secure-boot.yaml
```

3. In your cluster `SiteConfig` CR, specify the required values for `extraManifestPath` and `bootMode`:

   a. Enter the directory name in the `.spec.clusters.extraManifestPath` field, for example:

```
clusters:
  - clusterName: "example-cluster"
    extraManifestPath: sno-extra-manifest/
```

   b. Set the value for `.spec.clusters.nodes.bootMode` to `UEFISecureBoot`, for example:

```
nodes:
  - hostName: "ran.example.lab"
    bootMode: "UEFISecureBoot"
```

4. Deploy the cluster using the GitOps ZTP pipeline.

Verification

1. Open a remote shell to the deployed cluster, for example:

```
$ oc debug node/node-1.example.com
```

2. Verify that the `SecureBoot` feature is enabled:

```
sh-4.4# mokutil --sb-state
```

Example output

```
SecureBoot enabled
```

Installing the GitOps ZTP pipeline

The procedures in this section tell you how to complete the following tasks:

- Prepare the Git repository you need to host site configuration data.
- Configure the hub cluster for generating the required installation and policy custom resources (CR).
- Deploy the managed clusters using zero touch provisioning (ZTP).

Preparing the ZTP Git repository

Create a Git repository for hosting site configuration data. The zero touch provisioning (ZTP) pipeline requires read access to this repository.

Procedure

1. Create a directory structure with separate paths for the `SiteConfig` and `PolicyGenTemplate` custom resources (CR).
2. Export the `argocd` directory from the `ztp-site-generate` container image using the following commands:

```
$ podman pull quay.io/openshift-kni/ztp-site-generator:4.10.0
```

```
$ mkdir -p ./out
```

```
$ podman run --log-driver=none --rm quay.io/openshift-kni/ztp-site-
generator:4.10.0 extract /home/ztp --tar | tar x -C ./out
```

3. Check that the `out` directory contains the following subdirectories:
   - `out/extra-manifest` contains the source CR files that `SiteConfig` uses to generate extra manifest `configMap`.
   - `out/source-crs` contains the source CR files that `PolicyGenTemplate` uses to generate the Red Hat Advanced Cluster Management (RHACM) policies.
   - `out/argocd/deployment` contains patches and YAML files to apply on the hub cluster for use in the next step of this procedure.

- out/argocd/example contains the examples for SiteConfig and PolicyGenTemplate files that represent the recommended configuration.

The directory structure under out/argocd/example serves as a reference for the structure and content of your Git repository. The example includes SiteConfig and PolicyGenTemplate reference CRs for single-node, three-node, and standard clusters. Remove references to cluster types that you are not using. The following example describes a set of CRs for a network of single-node clusters:

```
example/
├── policygentemplates
│   ├── common-ranGen.yaml
│   ├── example-sno-site.yaml
│   ├── group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   ├── kustomization.yaml
│   └── ns.yaml
└── siteconfig
    ├── example-sno.yaml
    ├── KlusterletAddonConfigOverride.yaml
    └── kustomization.yaml
```

Keep SiteConfig and PolicyGenTemplate CRs in separate directories. Both the SiteConfig and PolicyGenTemplate directories must contain a kustomization.yaml file that explicitly includes the files in that directory.

This directory structure and the kustomization.yaml files must be committed and pushed to your Git repository. The initial push to Git should include the kustomization.yaml files. The SiteConfig (example-sno.yaml) and PolicyGenTemplate (common-ranGen.yaml, group-du-sno*.yaml, and example-sno-site.yaml) files can be omitted and pushed at a later time as required when deploying a site.

The KlusterletAddonConfigOverride.yaml file is only required if one or more SiteConfig CRs which make reference to it are committed and pushed to Git. See example-sno.yaml for an example of how this is used.

Preparing the hub cluster for ZTP

You can configure your hub cluster with a set of ArgoCD applications that generate the required installation and policy custom resources (CR) for each site based on a zero touch provisioning (ZTP) GitOps flow.

Prerequisites

- Openshift Cluster 4.8 or 4.9 as the hub cluster
- Red Hat Advanced Cluster Management (RHACM) Operator 2.3 or 2.4 installed on the hub cluster
- Red Hat OpenShift GitOps Operator 1.3 on the hub cluster

Procedure

1. Install the {cgu-operatpor-first}, which coordinates with any new sites added by ZTP and manages application of the `PolicyGenTemplate`-generated policies.
2. Prepare the ArgoCD pipeline configuration:
   a. Create a Git repository with the directory structure similar to the example directory. For more information, see "Preparing the ZTP Git repository".
   b. Configure access to the repository using the ArgoCD UI. Under **Settings** configure the following:
      - **Repositories** - Add the connection information. The URL must end in `.git`, for example, `https://repo.example.com/repo.git` and credentials.
      - **Certificates** - Add the public certificate for the repository, if needed.
   c. Modify the two ArgoCD Applications, `out/argocd/deployment/clusters-app.yaml` and `out/argocd/deployment/policies-app.yaml`, based on your Git repository:
      - Update the URL to point to the Git repository. The URL must end with `.git`, for example, `https://repo.example.com/repo.git`.
      - The `targetRevision` must indicate which Git repository branch to monitor.
      - The path should specify the path to the `SiteConfig` or `PolicyGenTemplate` CRs, respectively.
3. Apply the pipeline configuration to your hub cluster using the following command:

```
$ oc apply -k out/argocd/deployment
```

Deploying additional changes to clusters

Custom resources (CRs) that are deployed through the GitOps zero touch provisioning (ZTP) pipeline support two goals:

1. Deploying additional Operators to spoke clusters that are required by typical RAN DU applications running at the network far-edge.
2. Customizing the OpenShift Container Platform installation to provide a high performance platform capable of meeting the strict timing requirements in a minimal CPU budget.

If you require cluster configuration changes outside of the base GitOps ZTP pipeline configuration, there are three options:

Apply the additional configuration after the ZTP pipeline is complete

When the GitOps ZTP pipeline deployment is complete, the deployed cluster is ready for application workloads. At this point, you can install additional Operators and apply configurations specific to your requirements. Ensure that additional configurations do not negatively affect the performance of the platform or allocated CPU budget.

Add content to the ZTP library

The base source CRs that you deploy with the GitOps ZTP pipeline can be augmented with custom content as required.

Create extra manifests for the cluster installation

Extra manifests are applied during installation and makes the installation process more efficient.

Providing additional source CRs or modifying existing source CRs can significantly impact the performance or CPU profile of OpenShift Container Platform.

Additional resources

- See Adding new content to the GitOps ZTP pipeline for more information about adding or modifying existing source CRs in the `ztp-site-generate` container.
- See Customizing the ZTP GitOps pipeline with extra manifests for more information on adding extra manifests.

Adding new content to the GitOps ZTP pipeline

The source CRs in the GitOps ZTP site generator container provide a set of critical features and node tuning settings for RAN Distributed Unit (DU) applications. These are applied to the clusters that you deploy with ZTP. To add or modify existing source CRs in the `ztp-site-generate` container, rebuild the `ztp-site-generate` container and make it available to the hub cluster, typically from the disconnected registry associated with the hub cluster. Any valid OpenShift Container Platform CR can be added.

Perform the following procedure to add new content to the ZTP pipeline.

Procedure

1. Create a directory containing a Containerfile and the source CR YAML files that you want to include in the updated `ztp-site-generate` container, for example:

```
ztp-update/
├──── example-cr1.yaml
├──── example-cr2.yaml
└──── ztp-update.in
```

2. Add the following content to the `ztp-update.in` Containerfile:

```
FROM quay.io/openshift-kni/ztp-site-generator:4.10.0

ADD example-cr2.yaml
/kustomize/plugin/ran.openshift.io/v1/policygentemplate/source-crs/
ADD example-cr1.yaml
/kustomize/plugin/ran.openshift.io/v1/policygentemplate/source-crs/
```

3. Open a terminal at the `ztp-update/` folder and rebuild the container:

```
$ podman build -t ztp-site-generate-rhel8-custom:v4.10-custom-1
```

4. Push the built container image to your disconnected registry, for example:

```
$ podman push localhost/ztp-site-generate-rhel8-custom:v4.10-custom-1
registry.example.com:5000/ztp-site-generate-rhel8-custom:v4.10-custom-1
```

5. Patch the Argo CD instance on the hub cluster to point to the newly built container image:

```
$ oc patch -n openshift-gitops argocd openshift-gitops --type=json -p '[{"op":
"replace", "path":"/spec/repo/initContainers/0/image", "value":
"registry.example.com:5000/ztp-site-generate-rhel8-custom:v4.10-custom-1"} ]'
```

When the Argo CD instance is patched, the `openshift-gitops-repo-server` pod automatically restarts.

Verification

1. Verify that the new `openshift-gitops-repo-server` pod has completed initialization and that the previous repo pod is terminated:

```
$ oc get pods -n openshift-gitops | grep openshift-gitops-repo-server
```

Example output

```
openshift-gitops-server-7df86f9774-db682          1/1     Running           1
28s
```

You must wait until the new `openshift-gitops-repo-server` pod has completed initialization and the previous pod is terminated before the newly added container image content is available.

Additional resources

- Alternatively, you can patch the Argo CD instance as described in Preparing the hub cluster for ZTP by modifying `argocd-openshift-gitops-patch.json` with an updated `initContainer`

image before applying the patch file.

Customizing extra installation manifests in the ZTP GitOps pipeline

You can define a set of extra manifests for inclusion in the installation phase of the zero touch provisioning (ZTP) GitOps pipeline. These manifests are linked to the `SiteConfig` custom resources (CRs) and are applied to the cluster during installation. Including `MachineConfig` CRs at install time makes the installation process more efficient.

Prerequisites

- Create a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and be defined as a source repository for the Argo CD application.

Procedure

1. Create a set of extra manifest CRs that the ZTP pipeline uses to customize the cluster installs.
2. In your custom `/siteconfig` directory, create an `/extra-manifest` folder for your extra manifests. The following example illustrates a sample `/siteconfig` with `/extra-manifest` folder:

```
siteconfig
├──── site1-sno-du.yaml
├──── site2-standard-du.yaml
└──── extra-manifest
    └──── 01-example-machine-config.yaml
```

3. Add your custom extra manifest CRs to the `siteconfig/extra-manifest` directory.
4. In your `SiteConfig` CR, enter the directory name in the `extraManifestPath` field, for example:

```
clusters:
- clusterName: "example-sno"
  networkType: "OVNKubernetes"
  extraManifestPath: extra-manifest
```

5. Save the `SiteConfig` CRs and `/extra-manifest` CRs and push them to the site configuration repo.

The ZTP pipeline appends the CRs in the `/extra-manifest` directory to the default set of extra manifests during cluster provisioning.

Deploying a site

Use the following procedure to prepare the hub cluster for site deployment and initiate zero touch provisioning (ZTP) by pushing custom resources (CRs) to your Git repository.

Procedure

1. Create the required secrets for the site. These resources must be in a namespace with a name matching the cluster name. In `out/argocd/example/siteconfig/example-sno.yaml`, the cluster name and namespace is `example-sno`.

   Create the namespace for the cluster using the following commands:

   ```
   $ export CLUSTERNS=example-sno
   ```

   ```
   $ oc create namespace $CLUSTERNS
   ```

2. Create a pull secret for the cluster. The pull secret must contain all the credentials necessary for installing OpenShift Container Platform and all required Operators. In all of the example `SiteConfig` CRs, the pull secret is named `assisted-deployment-pull-secret`, as shown below:

   ```
   $ oc apply -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: assisted-deployment-pull-secret
     namespace: $CLUSTERNS
   type: kubernetes.io/dockerconfigjson
   data:
     .dockerconfigjson: $(base64 <pull-secret.json)
   EOF
   ```

3. Create a BMC authentication secret for each host you are deploying:

   ```
   $ oc apply -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: $(read -p 'Hostname: ' tmp; printf $tmp)-bmc-secret
     namespace: $CLUSTERNS
   type: Opaque
   data:
     username: $(read -p 'Username: ' tmp; printf $tmp | base64)
     password: $(read -s -p 'Password: ' tmp; printf $tmp | base64)
   EOF
   ```

   The secrets are referenced from the `SiteConfig` custom resource (CR) by name. The namespace must match the `SiteConfig` namespace.

4. Create a `SiteConfig` CR for your cluster in your local clone of the Git repository:

   a. Choose the appropriate example for your CR from the `out/argocd/example/siteconfig/`

folder. The folder includes example files for single node, three-node, and standard clusters:

- example-sno.yaml
- example-3node.yaml
- example-standard.yaml

b. Change the cluster and host details in the example file to match the type of cluster you want. The following file is a composite of the three files that explains the configuration of each cluster type:

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be
created under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10" ①
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
    networkType: "OVNKubernetes"
    clusterLabels: ②
      # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples in ../policygentemplates:
      # ../policygentemplates/common-ranGen.yaml will apply to all clusters
with 'common: true'
      common: true
      # ../policygentemplates/group-du-sno-ranGen.yaml will apply to all
clusters with 'group-du-sno: ""'
      group-du-sno: ""
      # ../policygentemplates/example-sno-site.yaml will apply to all
clusters with 'sites: "example-sno"'
      # Normally this should match or contain the cluster name so it only
applies to a single cluster
      sites : "example-sno"
    clusterNetwork:
      - cidr: 1001:1::/48
        hostPrefix: 64
    machineNetwork: ③
      - cidr: 1111:2222:3333:4444::/64
      # For 3-node and standard clusters with static IPs, the API and
Ingress IPs must be configured here
    apiVIP: 1111:2222:3333:4444::1:1 ④
```

```
    ingressVIP: 1111:2222:3333:4444::1:2 ⑤

    serviceNetwork:
      - 1001:2::/112
    additionalNTPSources:
      - 1111:2222:3333:4444::2
    nodes:
      - hostName: "example-node1.example.com" ⑥
        role: "master"
        bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/Syste
m.Embedded.1" ⑦
        bmcCredentialsName:
          name: "example-node1-bmh-secret" ⑧
        bootMACAddress: "AA:BB:CC:DD:EE:11"
        bootMode: "UEFI"
        rootDeviceHints:
          hctl: '0:1:0'
        cpuset: "0-1,52-53"
        nodeNetwork: ⑨
          interfaces:
            - name: eno1
              macAddress: "AA:BB:CC:DD:EE:11"
          config:
            interfaces:
              - name: eno1
                type: ethernet
                state: up
                macAddress: "AA:BB:CC:DD:EE:11"
                ipv4:
                  enabled: false
                ipv6:
                  enabled: true
                  address:
                  - ip: 1111:2222:3333:4444::1:1
                    prefix-length: 64
            dns-resolver:
              config:
                search:
                - example.com
                server:
                - 1111:2222:3333:4444::2
            routes:
              config:
              - destination: ::/0
                next-hop-interface: eno1
                next-hop-address: 1111:2222:3333:4444::1
                table-id: 254
```

① Applies to all cluster types. The value must match an image set available on the hub cluster. To see the list of supported versions on your hub, run `oc get`

`clusterimagesets`.

② Applies to all cluster types. These values must correspond to the `PolicyGenTemplate` labels that you define in a later step.

③ Applies to single node clusters. The value defines the cluster network sections for a single node deployment.

④ Applies to three-node and standard clusters. The value defines the cluster network sections.

⑤ Applies to three-node and standard clusters. The value defines the cluster network sections.

⑥ Applies to all cluster types. For single node deployments, define one host. For three-node deployments, define three hosts. For standard deployments, define three hosts with `role: master` and two or more hosts defined with `role: worker`.

⑦ Applies to all cluster types. Specifies the BMC address.

⑧ Applies to all cluster types. Specifies the BMC credentials.

⑨ Applies to all cluster types. Specifies the network settings for the node.

c. You can inspect the default set of extra-manifest `MachineConfig` CRs in `out/argocd/extra-manifest`. It is automatically applied to the cluster when it is installed.

Optional: To provision additional install-time manifests on the provisioned cluster, create a directory in your Git repository, for example, `sno-extra-manifest/`, and add your custom manifest CRs to this directory. If your `SiteConfig.yaml` refers to this directory in the `extraManifestPath` field, any CRs in this referenced directory are appended to the default set of extra manifests.

5. Add the `SiteConfig` CR to the `kustomization.yaml` file in the `generators` section, similar to the example shown in `out/argocd/example/siteconfig/kustomization.yaml`.
6. Commit your `SiteConfig` CR and associated `kustomization.yaml` in your Git repository.
7. Push your changes to the Git repository. The ArgoCD pipeline detects the changes and begins the site deployment. You can push the changes to the `SiteConfig` CR and the `PolicyGenTemplate` CR simultaneously.

The `SiteConfig` CR creates the following CRs on the hub cluster:

- `Namespace` - Unique per site
- `AgentClusterInstall`
- `BareMetalHost` - One per node
- `ClusterDeployment`
- `InfraEnv`
- `NMStateConfig` - One per node
- `ExtraManifestsConfigMap` - Extra manifests. The additional manifests include workload partitioning, chronyd, mountpoint hiding, sctp enablement, and more.

- ManagedCluster
- KlusterletAddonConfig

GitOps ZTP and Cluster Group Upgrades Operator (CGUO)

GitOps zero touch provisioning (ZTP) generates installation and configuration CRs from manifests stored in Git. These artifacts are applied to a centralized hub cluster where Red Hat Advanced Cluster Management (RHACM), assisted installer service, and the Cluster Group Upgrades Operator (CGUO) use the CRs to install and configure the spoke cluster. The configuration phase of the ZTP pipeline uses the CGUO to orchestrate the application of the configuration CRs to the cluster. There are several key integration points between GitOps ZTP and the CGUO.

Inform policies

By default, GitOps ZTP creates all policies with a remediation action of `inform`. These policies cause RHACM to report on compliance status of clusters relevant to the policies but does not apply the desired configuration. During the ZTP installation, the CGUO steps through the created `inform` policies, creates a copy for the target spoke cluster(s) and changes the remediation action of the copy to `enforce`. This pushes the configuration to the spoke cluster. Outside of the ZTP phase of the cluster lifecycle, this setup allows changes to be made to policies without the risk of immediately rolling those changes out to all affected spoke clusters in the network. You can control the timing and the set of clusters that are remediated using the CGUO.

Automatic creation of ClusterGroupUpgrade CRs

The CGUO monitors the state of all `ManagedCluster` CRs on the hub cluster. Any `ManagedCluster` CR which does not have a `ztp-done` label applied, including newly created `ManagedCluster` CRs, causes the CGUO to automatically create a `ClusterGroupUpgrade` CR with the following characteristics:

- The `ClusterGroupUpgrade` CR is created and enabled in the `ztp-install` namespace.
- `ClusterGroupUpgrade` CR has the same name as the `ManagedCluster` CR.
- The cluster selector includes only the cluster associated with that `ManagedCluster` CR.
- The set of managed policies includes all policies that RHACM has bound to the cluster at the time the `ClusterGroupUpgrade` is created.
- Pre-caching is disabled.
- Timeout set to 4 hours (240 minutes).

The automatic creation of an enabled `ClusterGroupUpgrade` ensures that initial zero-touch deployment of clusters proceeds without the need for user intervention. Additionally, the automatic creation of a `ClusterGroupUpgrade` CR for any `ManagedCluster` without the `ztp-done` label allows a failed ZTP installation to be restarted by simply deleting the `ClusterGroupUpgrade` CR for the cluster.

Waves

Each policy generated from a `PolicyGenTemplate` CR includes a `ztp-deploy-wave` annotation. This annotation is based on the same annotation from each CR which is included in that policy. The wave annotation is used to order the policies in the auto-generated `ClusterGroupUpgrade` CR.

All CRs in the same policy must have the same setting for the `ztp-deploy-wave` annotation. The default value of this annotation for each CR can be overridden in the `PolicyGenTemplate`.

The CGUO applies the configuration policies in the order specified by the wave annotations. The CGUO waits for each policy to be compliant before moving to the next policy. It is important to ensure that the wave annotation for each CR takes into account any prerequisites for those CRs to be applied to the cluster. For example, an Operator must be installed before, or concurrently with, the configuration for the Operator. The default wave value for each CR takes these prerequisites into account.

Multiple CRs and policies can share the same wave number. Having fewer policies can result in faster deployments and lower CPU usage. It is a best practice to group many CRs into relatively few waves.

Phase labels

The `ClusterGroupUpgrade` CR is automatically created and includes directives to annotate the `ManagedCluster` CR with labels at the start and end of the ZTP process.

When ZTP configuration post-installation commences, the `ManagedCluster` has the `ztp-running` label applied. When all policies are remediated to the cluster and are fully compliant, these directives cause the CGUO to remove the `ztp-running` label and apply the `ztp-done` label.

For deployments which make use of the `informDuValidator` policy, the `ztp-done` label is applied when the cluster is fully ready for deployment of applications. This includes all reconciliation and resulting effects of the ZTP applied configuration CRs.

Linked CRs

The automatically created `ClusterGroupUpgrade` CR has the owner reference set as the `ManagedCluster` from which it was derived. This reference ensures that deleting the `ManagedCluster` CR causes the instance of the `ClusterGroupUpgrade` to be deleted along with any supporting resources.

Monitoring deployment progress

The ArgoCD pipeline uses the `SiteConfig` and `PolicyGenTemplate` CRs in Git to generate the cluster configuration CRs and RHACM policies and then sync them to the hub. You can monitor the progress of this synchronization can be monitored in the ArgoCD dashboard.

Procedure

When the synchronization is complete, the installation generally proceeds as follows:

1. The Assisted Service Operator installs OpenShift Container Platform on the cluster. You can monitor the progress of cluster installation from the RHACM dashboard or from the command line:

```
$ export CLUSTER=<clusterName>
```

```
$ oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath
='{.status.conditions[?(@.type=="Completed")]}' | jq
```

```
$ curl -sk $(oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath
='{.status.debugInfo.eventsURL}')  | jq '.[-2,-1]'
```

2. The Cluster Group Upgrades Operator (CGUO) applies the configuration policies that are bound to the cluster.

   After the cluster installation is complete and the cluster becomes `Ready`, a `ClusterGroupUpgrade` CR corresponding to this cluster, with a list of ordered policies defined by the `ran.openshift.io/ztp-deploy-wave annotations`, is automatically created by the CGUO. The cluster's policies are applied in the order listed in `ClusterGroupUpgrade` CR. You can monitor the high-level progress of configuration policy reconciliation using the following commands:

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath
='{.status.conditions[?(@.type=="Ready")]}'
```

3. You can monitor the detailed policy compliant status using the RHACM dashboard or the command line:

```
$ oc get policies -n $CLUSTER
```

The final policy that becomes compliant is the one defined in the `*-du-validator-policy` policies. This policy, when compliant on a cluster, ensures that all cluster configuration, Operator installation, and Operator configuration is complete.

After all policies become complaint, the `ztp-done` label is added to the cluster, indicating the entire ZTP pipeline is complete for the cluster.

Indication of done for ZTP installations

Zero touch provisioning (ZTP) simplifies the process of checking the ZTP installation status for a

cluster. The ZTP status moves through three phases: cluster installation, cluster configuration, and ZTP done.

Cluster installation phase

The cluster installation phase is shown by the `ManagedCluster` CR `ManagedClusterJoined` condition. If the `ManagedCluster` CR does not have this condition, or the condition is set to `False`, the cluster is still in the installation phase. Additional details about installation are available from the `AgentClusterInstall` and `ClusterDeployment` CRs. For more information, see "Troubleshooting GitOps ZTP".

Cluster configuration phase

The cluster configuration phase is shown by a `ztp-running` label applied the the `ManagedCluster` CR for the cluster.

ZTP done

Cluster installation and configuration is complete in the ZTP done phase. This is shown by the removal of the `ztp-running` label and addition of the `ztp-done` label to the `ManagedCluster` CR. The `ztp-done` label shows that the configuration has been applied and the baseline DU configuration has completed cluster tuning.

The transition to the ZTP done state is conditional on the compliant state of a Red Hat Advanced Cluster Management (RHACM) static validator inform policy. This policy captures the existing criteria for a completed installation and validates that it moves to a compliant state only when ZTP provisioning of the spoke cluster is complete.

The validator inform policy ensures the configuration of the distributed unit (DU) cluster is fully applied and Operators have completed their initialization. The policy validates the following:

- The target `MachineConfigPool` contains the expected entries and has finished updating. All nodes are available and not degraded.
- The SR-IOV Operator has completed initialization as indicated by at least one `SriovNetworkNodeState` with `syncStatus: Succeeded`.
- The PTP Operator daemon set exists.

  The policy captures the existing criteria for a completed installation and validates that it moves to a compliant state only when ZTP provisioning of the spoke cluster is complete.

  The validator inform policy is included in the reference group `PolicyGenTemplate` CRs. For reliable indication of the ZTP done state, this validator inform policy must be included in the ZTP pipeline.

Creating a validator inform policy

Use the following procedure to create a validator inform policy that provides an indication of when the zero touch provisioning (ZTP) installation and configuration of the deployed cluster is complete. This policy can be used for deployments of single node clusters, three-node clusters, and standard clusters.

Procedure

1. Create a stand-alone `PolicyGenTemplate` custom resource (CR) that contains the source file `validatorCRs/informDuValidator.yaml`. You only need one stand-alone `PolicyGenTemplate` CR for each cluster type.

   Single node clusters

```
group-du-sno-validator-ranGen.yaml
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno-validator" ①
  namespace: "ztp-group" ②
spec:
  bindingRules:
    group-du-sno: "" ③
  bindingExcludedRules:
    ztp-done: "" ④
  mcp: "master" ⑤
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform ⑥
      policyName: "du-policy" ⑦
```

   Three-node clusters

```
group-du-3node-validator-ranGen.yaml
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-3node-validator" ①
  namespace: "ztp-group" ②
spec:
  bindingRules:
    group-du-3node: "" ③
  bindingExcludedRules:
    ztp-done: "" ④
  mcp: "master" ⑤
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform ⑥
      policyName: "du-policy" ⑦
```

Standard clusters

```
group-du-standard-validator-ranGen.yaml
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-standard-validator" ①
  namespace: "ztp-group" ②
spec:
  bindingRules:
    group-du-standard: "" ③
  bindingExcludedRules:
    ztp-done: "" ④
  mcp: "worker" ⑤
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform ⑥
      policyName: "du-policy" ⑦
```

① The name of `PolicyGenTemplates` object. This name is also used as part of the names for the `placementBinding`, `placementRule`, and `policy` that are created in the requested `namespace`.

② This value should match the `namespace` used in the group `PolicyGenTemplates`.

③ The `group-du-*` label defined in `bindingRules` must exist in the `SiteConfig` files.

④ The label defined in `bindingExcludedRules` must be`ztp-done:`. The `ztp-done` label is used in coordination with the Cluster Group Upgrades Operator (CGUO).

⑤ `mcp` defines the `MachineConfigPool` object that is used in the source file `validatorCRs/informDuValidator.yaml`. It should be `master` for single node and three-node cluster deployments and `worker` for standard cluster deployments.

⑥ Optional. The default value is `inform`.

⑦ This value is used as part of the name for the generated RHACM policy. The generated validator policy for the single node example is named `group-du-sno-validator-du-policy`.

2. Push the files to the ZTP Git repository.

Querying the policy compliance status for each cluster

After you have created the validator inform policies for your clusters and pushed them to the the zero touch provisioning (ZTP) Git repository, you can check the status of each cluster for policy compliance.

Procedure

1. To query the status of the spoke clusters, use either the Red Hat Advanced Cluster Management (RHACM) web console or the CLI:

   To query status from the RHACM web console, perform the following actions:

   a. Click **Governance** **Find policies**.

b. Search for **du-validator-policy**.

c. Click into the policy.

- To query status using the CLI, run the following command:

```
$ oc get policies du-validator-policy -n <namespace_for_common> -o
jsonpath={'.status.status'} | jq
```

When all of the policies including the validator inform policy applied to the cluster become compliant, ZTP installation and configuration for this cluster is complete.

2. To query the cluster violation/compliant status from the ACM web console, click **Governance** **Cluster violations**.
3. Check the validator policy compliant status for a cluster using the following commands:

a. Export the cluster name:

```
$ export CLUSTER=<cluster_name>
```

b. Get the policy:

```
$ oc get policies -n $CLUSTER | grep <validator_policy_name>
```

Alternatively, you can use the following command:

```
$ oc get policies -n <namespace-for-group> <validatorPolicyName> -o jsonpath
="{.status.status[?(@.clustername=='$CLUSTER')]}" | jq
```

After the `*-validator-du-policy` RHACM policy becomes compliant for the cluster, the validator policy is unbound for this cluster and the `ztp-done` label is added to the cluster. This acts as a persistent indicator that the whole ZTP pipeline has completed for the cluster.

Troubleshooting GitOps ZTP

The ArgoCD pipeline uses the `SiteConfig` and `PolicyGenTemplate` custom resources (CRs) from Git to generate the cluster configuration CRs and Red Hat Advanced Cluster Management (RHACM) policies. Use the following steps to troubleshoot issues that might occur during this process.

file// Module included in the following assemblies:

Validating the generation of installation CRs

The GitOps zero touch provisioning (ZTP) infrastructure generates a set of installation CRs on

the hub cluster in response to a SiteConfig CR pushed to your Git repository. You can check that the installation CRs were created by using the following command:

```
$ oc get AgentClusterInstall -n <cluster_name>
```

If no object is returned, use the following procedure to troubleshoot the ArgoCD pipeline flow from SiteConfig files to the installation CRs.

Procedure

1. Verify that the SiteConfig→ManagedCluster was generated to the hub cluster:

   ```
   $ oc get managedcluster
   ```

2. If the SiteConfig ManagedCluster is missing, see if the clusters application failed to synchronize the files from the Git repository to the hub:

   ```
   $ oc describe -n openshift-gitops application clusters
   ```

3. Check for Status: Conditions: to view the error logs. For example, setting an invalid value for extraManifestPath: in the siteConfig file raises an error as shown below:

   ```
   Status:
     Conditions:
       Last Transition Time:  2021-11-26T17:21:39Z
       Message:               rpc error: code = Unknown desc = `kustomize build
   /tmp/https___git.com/ran-sites/siteconfigs/ --enable-alpha-plugins` failed exit
   status 1: 2021/11/26 17:21:40 Error could not create extra-manifest
   ranSite1.extra-manifest3 stat extra-manifest3: no such file or directory
   2021/11/26 17:21:40 Error: could not build the entire SiteConfig defined by
   /tmp/kust-plugin-config-913473579: stat extra-manifest3: no such file or
   directory
   Error: failure in plugin configured via /tmp/kust-plugin-config-913473579; exit
   status 1: exit status 1
       Type:  ComparisonError
   ```

4. Check for Status: Sync:. If there are log errors, Status: Sync: could indicate an Unknown error:

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  clusters-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:             sites-config
        Repo URL:         https://git.com/ran-sites/siteconfigs/.git
        Target Revision:  master
    Status:               Unknown
```

Validating the generation of configuration policy CRs

Policy custom resources (CRs) are generated in the same namespace as the `PolicyGenTemplate` from which they are created. The same troubleshooting flow applies to all policy CRs generated from a `PolicyGenTemplate` regardless of whether they are `ztp-common`, `ztp-group`, or `ztp-site` based, as shown using the following commands:

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

The expected set of policy-wrapped CRs should be displayed.

If the policies failed synchronization, use the following troubleshooting steps.

Procedure

1. To display detailed information about the policies, run the following command:

```
$ oc describe -n openshift-gitops application policies
```

2. Check for `Status: Conditions:` to show the error logs. For example, setting an invalid `sourceFile→fileName:` generates the error shown below:

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit
status 1: 2021/11/26 17:21:40 Error could not find test.yaml under source-crs/:
no such file or directory
Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit
status 1: exit status 1
    Type:  ComparisonError
```

3. Check for `Status: Sync:`. If there are log errors at `Status: Conditions:`, the `Status: Sync:` shows `Unknown` or `Error`:

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  policies-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:             policies
        Repo URL:         https://git.com/ran-sites/policies/.git
        Target Revision:  master
    Status:               Error
```

4. When Red Hat Advanced Cluster Management (RHACM) recognizes that policies apply to a `ManagedCluster` object, the policy CR objects are applied to the cluster namespace. Check to see if the policies were copied to the cluster namespace:

```
$ oc get policy -n $CLUSTER
```

Example output:

```
NAME                                    REMEDIATION ACTION   COMPLIANCE
STATE   AGE
ztp-common.common-config-policy         inform               Compliant
13d
ztp-common.common-subscriptions-policy  inform               Compliant
13d
ztp-group.group-du-sno-config-policy    inform               Compliant
13d
Ztp-group.group-du-sno-validator-du-policy   inform          Compliant
13d
ztp-site.example-sno-config-policy      inform               Compliant
13d
```

RHACM copies all applicable policies into the cluster namespace. The copied policy names have the format: `<policyGenTemplate.Namespace>.<policyGenTemplate.Name>-<policyName>`.

5. Check the placement rule for any policies not copied to the cluster namespace. The `matchSelector` in the `PlacementRule` for those policies should match labels on the `ManagedCluster` object:

```
$ oc get placementrule -n $NS
```

6. Note the `PlacementRule` name appropriate for the missing policy, common, group, or site, using the following command:

```
$ oc get placementrule -n $NS <placementRuleName> -o yaml
```

   - The status-decisions should include your cluster name.
   - The key-value pair of the `matchSelector` in the spec must match the labels on your managed cluster.

7. Check the labels on the `ManagedCluster` object using the following command:

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. Check to see which policies are compliant using the following command:

```
$ oc get policy -n $CLUSTER
```

   If the `Namespace`, `OperatorGroup`, and `Subscription` policies are compliant but the Operator configuration policies are not, it is likely that the Operators did not install on the spoke cluster. This causes the Operator configuration policies to fail to apply because the CRD is not yet applied to the spoke.

Restarting policies reconciliation

Use the following procedure to restart policies reconciliation in the event of unexpected compliance issues. This procedure is required when the `ClusterGroupUpgrade` CR has timed out.

Procedure

1. A `ClusterGroupUpgrade` CR is generated in the namespace `ztp-install` by the Cluster Group Upgrades Operator (CGUO) after the managed spoke cluster becomes `Ready`:

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. If there are unexpected issues and the policies fail to become complaint within the configured timeout (the default is 4 hours), the status of the `ClusterGroupUpgrade` CR shows `UpgradeTimedOut`:

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath
='{.status.conditions[?(@.type=="Ready")]}'
```

3. A `ClusterGroupUpgrade` CR in the `UpgradeTimedOut` state automatically restarts its policy reconciliation every hour. If you have changed your policies, you can start a retry immediately by deleting the existing `ClusterGroupUpgrade` CR. This triggers the automatic creation of a new `ClusterGroupUpgrade` CR that begins reconciling the policies immediately:

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

Note that when the `ClusterGroupUpgrade` CR completes with status `UpgradeCompleted` and the managed spoke cluster has the label `ztp-done` applied, you can make additional configuration changes using `PolicyGenTemplate`. Deleting the existing `ClusterGroupUpgrade` CR will not make the CGUO generate a new CR.

At this point, ZTP has completed its interaction with the cluster and any further interactions should be treated as an upgrade.

Additional resources

- For information about using CGUO to construct your own `ClusterGroupUpgrade` CR, see About the ClusterGroupUpgrade CR.

Site cleanup

Remove a site and the associated installation and configuration policy CRs by removing the `SiteConfig` and `PolicyGenTemplate` file names from the `kustomization.yaml` file. When you run the ZTP pipeline again, the generated CRs are removed. If you want to permanently remove a site, you should also remove the `SiteConfig` and site-specific `PolicyGenTemplate` files from the Git repository. If you want to remove a site temporarily, for example when redeploying a site, you can leave the `SiteConfig` and site-specific `PolicyGenTemplate` CRs in the Git repository.

After removing the `SiteConfig` file, if the corresponding clusters remain in the detach process, check Red Hat Advanced Cluster Management (RHACM) for information about cleaning up the detached managed cluster.

Additional resources

- For information about removing a cluster, see Removing a cluster from management.

Removing obsolete content

If a change to the `PolicyGenTemplate` file configuration results in obsolete policies, for example, policies are renamed, use the following procedure to remove those policies in an automated way.

Procedure

1. Remove the affected `PolicyGenTemplate` files from the Git repository, commit and push to the remote repository.
2. Wait for the changes to synchronize through the application and the affected policies to be removed from the hub cluster.
3. Add the updated `PolicyGenTemplate` files back to the Git repository, and then commit and push to the remote repository.

Note that removing the zero touch provisioning (ZTP) distributed unit (DU) profile policies from the Git repository, and as a result also removing them from the hub cluster, does not affect any configuration of the managed spoke clusters. Removing a policy from the hub cluster does not delete it from the spoke cluster and the CRs managed by that policy.

As an alternative, after making changes to `PolicyGenTemplate` files that result in obsolete policies, you can remove these policies from the hub cluster manually. You can delete policies from the RHACM console using the **Governance** tab or by using the following command:

```
$ oc delete policy -n <namespace> <policyName>
```

Tearing down the pipeline

If you need to remove the ArgoCD pipeline and all generated artifacts follow this procedure:

Procedure

1. Detach all clusters from RHACM.
2. Delete the `kustomization.yaml` file in the `deployment` directory using the following command:

```
$ oc delete -k out/argocd/deployment
```

Upgrading GitOps ZTP

You can upgrade the Gitops zero touch provisioning (ZTP) infrastructure independently from the underlying cluster, Red Hat Advanced Cluster Management (RHACM), and OpenShift Container Platform version running on the spoke clusters. This procedure guides you through the upgrade process to avoid impact on the spoke clusters. However, any changes to the content or settings of policies, including adding recommended content, results in changes that must be rolled out and reconciled to the spoke clusters.

Prerequisites

- This procedure assumes that you have a fully operational hub cluster running the earlier version of the GitOps ZTP infrastructure.

Procedure

At a high level, the strategy for upgrading the GitOps ZTP infrastructure is:

1. Label all existing clusters with the `ztp-done` label.
2. Stop the ArgoCD applications.
3. Install the new tooling.
4. Update required content and optional changes in the Git repository.
5. Update and restart the application configuration.

Preparing for the upgrade

Use the following procedure to prepare your site for the GitOps zero touch provisioning (ZTP) upgrade.

Procedure

1. Obtain the latest version of the GitOps ZTP container from which you can extract a set of custom resources (CRs) used to configure the GitOps operator on the hub cluster for use in the GitOps ZTP solution.
2. Extract the `argocd/deployment` directory using the following commands:

```
$ mkdir -p ./out
```

```
$ podman run --log-driver=none --rm quay.io/openshift-kni/ztp-site-
generator:4.10.0 extract /home/ztp --tar | tar x -C ./out
```

   The `/out` directory contains the following subdirectories:

   - `out/extra-manifest`: contains the source CR files that the `SiteConfig` CR uses to generate the extra manifest `configMap`.
   - `out/source-crs`: contains the source CR files that the `PolicyGenTemplate` CR uses to generate the Red Hat Advanced Cluster Management (RHACM) policies.
   - `out/argocd/deployment`: contains patches and YAML files to apply on the hub cluster for use in the next step of this procedure.
   - `out/argocd/example`: contains example `SiteConfig` and `PolicyGenTemplate` files that represent the recommended configuration.

3. Update the `clusters-app.yaml` and `policies-app.yaml` files to reflect the name of your applications and the URL, branch, and path for your Git repository.

If the upgrade includes changes to policies that may result in obsolete policies, these policies should be removed prior to performing the upgrade.

Labeling the existing clusters

To ensure that existing clusters remain untouched by the tooling updates, all existing managed clusters must be labeled with the `ztp-done` label.

Procedure

1. Find a label selector that lists the managed clusters that were deployed with zero touch provisioning (ZTP), such as `local-cluster!=true`:

```
$ oc get managedcluster -l 'local-cluster!=true'
```

2. Ensure that the resulting list contains all the managed clusters that were deployed with ZTP, and then use that selector to add the `ztp-done` label:

```
$ oc label managedcluster -l 'local-cluster!=true' ztp-done=
```

Stopping the existing GitOps ZTP applications

Removing the existing applications ensures that any changes to existing content in the Git repository are not rolled out until the new version of the tooling is available.

Use the application files from the `deployment` directory. If you used custom names for the applications, update the names in these files first.

Procedure

1. Perform a non-cascaded delete on the `clusters` application to leave all generated resources in place:

```
$ oc delete -f out/argocd/deployment/clusters-app.yaml
```

2. Perform a cascaded delete on the `policies` application to remove all previous policies:

```
$ oc patch -f policies-app.yaml -p '{"metadata": {"finalizers": ["resources-finalizer.argocd.argoproj.io"]}}' --type merge
```

```
$ oc delete -f out/argocd/deployment/policies-app.yaml
```

Cluster Group Upgrades Operator (CGUO)

Install the Cluster Group Upgrades Operator (CGUO) on the hub cluster.

Additional resources

- For information about the Cluster Group Upgrades Operator (CGUO), see About the CGUO

configuration.

Required changes to the Git repository

When upgrading from an earlier release to OpenShift Container Platform 4.10, additional requirements are placed on the contents of the Git repository. Existing content in the repository must be updated to reflect these changes.

- Changes to `PolicyGenTemplate` files:

  All `PolicyGenTemplate` files must be created in a `Namespace` prefixed with `ztp`. This ensures that the GitOps zero touch provisioning (ZTP) application is able to manage the policy CRs generated by GitOps ZTP without conflicting with the way Red Hat Advanced Cluster Management (RHACM) manages the policies internally.

- Remove the `pre-sync.yaml` and `post-sync.yaml` files:

  This step is optional but recommended. When the `kustomization.yaml` files are added, the `pre-sync.yaml` and `post-sync.yaml` files are no longer used. They must be removed to avoid confusion and can potentially cause errors if kustomization files are inadvertantly removed. Note that there is a set of `pre-sync.yaml` and `post-sync.yaml` files under both the `SiteConfig` and `PolicyGenTemplate` trees.

- Add the `kustomization.yaml` file to the repository:

  All `SiteConfig` and `PolicyGenTemplate` CRs must be included in a `kustomization.yaml` file under their respective directory trees. For example:

```
├──── policygentemplates
│     ├──── site1-ns.yaml
│     ├──── site1.yaml
│     ├──── site2-ns.yaml
│     ├──── site2.yaml
│     ├──── common-ns.yaml
│     ├──── common-ranGen.yaml
│     ├──── group-du-sno-ranGen-ns.yaml
│     ├──── group-du-sno-ranGen.yaml
│     └──── kustomization.yaml
└──── siteconfig
      ├──── site1.yaml
      ├──── site2.yaml
      └──── kustomization.yaml
```

The files listed in the `generator` sections must contain either `SiteConfig` or `PolicyGenTemplate` CRs only. If your existing YAML files contain other CRs, for example, `Namespace`, these other CRs must be pulled out into separate files and listed in the `resources` section.

The `PolicyGenTemplate` kustomization file must contain all `PolicyGenTemplate` YAML files in

the generator section and Namespace CRs in the resources section. For example:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- common-ranGen.yaml
- group-du-sno-ranGen.yaml
- site1.yaml
- site2.yaml

resources:
- common-ns.yaml
- group-du-sno-ranGen-ns.yaml
- site1-ns.yaml
- site2-ns.yaml
```

The SiteConfig kustomization file must contain all SiteConfig YAML files in the generator section and any other CRs in the resources:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- site1.yaml
- site2.yaml
```

- Review and incorporate recommended changes

  Each release may include additional recommended changes to the configuration applied to deployed clusters. Typically these changes result in lower CPU use by the OpenShift platform, additional features, or improved tuning of the platform.

  Review the reference SiteConfig and PolicyGenTemplate CRs applicable to the types of cluster in your network. These examples can be found in the argocd/example directory extracted from the GitOps ZTP container.

Installing the new GitOps ZTP applications

Using the extracted argocd/deployment directory, and after ensuring that the applications point to your Git repository, apply the full contents of the deployment directory. Applying the full contents of the directory ensures that all necessary resources for the applications are correctly configured.

Procedure

- Apply the contents of the argocd/deployment directory using the following command:

```
$ oc apply -k out/argocd/deployment
```

Roll out the configuration changes

If any configuration changes were included in the upgrade due to implementing recommended changes, the upgrade process results in a set of policy CRs on the hub cluster in the `Non-Compliant` state. As of the OpenShift Container Platform 4.10 release, these policies are set to `inform` mode and are not pushed to the spoke clusters without an additional step by the user. This ensures that potentially disruptive changes to the clusters can be managed in terms of when the changes are made, for example, during a maintenance window, and how many clusters are updated concurrently.

To roll out the changes, create one or more `ClusterGroupUpgrade` CRs as detailed in the CGUO documentation. The CR must contain the list of `Non-Compliant` policies that you want to push out to the spoke clusters as well as a list or selector of which clusters should be included in the update.

Additional resources

- For information about creating `ClusterGroupUpgrade` CRs, see About the auto-created ClusterGroupUpgrade CR for ZTP.

Manually install a single managed cluster

This procedure tells you how to manually create and deploy a single managed cluster. If you are creating multiple clusters, perhaps hundreds, use the `SiteConfig` method described in “Creating ZTP custom resources for multiple managed clusters”.

Prerequisites

- Enable the Assisted Installer service.
- Ensure network connectivity:
  - The container within the hub must be able to reach the Baseboard Management Controller (BMC) address of the target bare-metal host.
  - The managed cluster must be able to resolve and reach the hub’s API `hostname` and `*.app` hostname. Here is an example of the hub’s API and `*.app` hostname:

    ```
    console-openshift-console.apps.hub-cluster.internal.domain.com
    api.hub-cluster.internal.domain.com
    ```

  - The hub must be able to resolve and reach the API and `*.app` hostname of the managed cluster. Here is an example of the managed cluster’s API and `*.app` hostname:

    ```
    console-openshift-console.apps.sno-managed-cluster-1.internal.domain.com
    api.sno-managed-cluster-1.internal.domain.com
    ```

- A DNS server that is IP reachable from the target bare-metal host.

- A target bare-metal host for the managed cluster with the following hardware minimums:
  - 4 CPU or 8 vCPU
  - 32 GiB RAM
  - 120 GiB disk for root file system

- When working in a disconnected environment, the release image must be mirrored. Use this command to mirror the release image:

```
$ oc adm release mirror -a <pull_secret.json>
--from=quay.io/openshift-release-dev/ocp-release:{{
mirror_version_spoke_release }}
--to={{ provisioner_cluster_registry }}/ocp4 --to-release-image={{
provisioner_cluster_registry }}/ocp4:{{ mirror_version_spoke_release }}
```

- You mirrored the ISO and `rootfs` used to generate the spoke cluster ISO to an HTTP server and configured the settings to pull images from there.

  The images must match the version of the `ClusterImageSet`. To deploy a 4.9.0 version, the `rootfs` and ISO must be set at 4.9.0.

Procedure

1. Create a `ClusterImageSet` for each specific cluster version that needs to be deployed. A `ClusterImageSet` has the following format:

```
apiVersion: hive.openshift.io/v1
kind: ClusterImageSet
metadata:
  name: openshift-4.9.0-rc.0 ①
spec:
   releaseImage: quay.io/openshift-release-dev/ocp-release:4.9.0-x86_64 ②
```

   ① The descriptive version that you want to deploy.

   ② The location of the specific release image to deploy.

2. Create the `Namespace` definition for the managed cluster:

```
apiVersion: v1
kind: Namespace
metadata:
     name: <cluster_name> ①
     labels:
        name: <cluster_name> ①
```

   ① The name of the managed cluster to provision.

3. Create the `BMC Secret` custom resource:

```
apiVersion: v1
data:
  password: <bmc_password> ①
  username: <bmc_username> ②
kind: Secret
metadata:
  name: <cluster_name>-bmc-secret
  namespace: <cluster_name>
type: Opaque
```

① The password to the target bare-metal host. Must be base-64 encoded.

② The username to the target bare-metal host. Must be base-64 encoded.

4. Create the `Image Pull Secret` custom resource:

```
apiVersion: v1
data:
  .dockerconfigjson: <pull_secret> ①
kind: Secret
metadata:
  name: assisted-deployment-pull-secret
  namespace: <cluster_name>
type: kubernetes.io/dockerconfigjson
```

① The OpenShift Container Platform pull secret. Must be base-64 encoded.

5. Create the `AgentClusterInstall` custom resource:

```
apiVersion: extensions.hive.openshift.io/v1beta1
kind: AgentClusterInstall
metadata:
  # Only include the annotation if using OVN, otherwise omit the annotation
  annotations:
    agent-install.openshift.io/install-config-overrides:
'{"networking":{"networkType":"OVNKubernetes"}}'
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterDeploymentRef:
    name: <cluster_name>
  imageSetRef:
    name: <cluster_image_set> ①
  networking:
    clusterNetwork:
    - cidr: <cluster_network_cidr> ②
      hostPrefix: 23
    machineNetwork:
    - cidr: <machine_network_cidr> ③
    serviceNetwork:
    - <service_network_cidr> ④
  provisionRequirements:
    controlPlaneAgents: 1
    workerAgents: 0
  sshPublicKey: <public_key> ⑤
```

① The name of the `ClusterImageSet` custom resource used to install OpenShift Container Platform on the bare-metal host.

② A block of IPv4 or IPv6 addresses in CIDR notation used for communication among cluster nodes.

③ A block of IPv4 or IPv6 addresses in CIDR notation used for the target bare-metal host external communication. Also used to determine the API and Ingress VIP addresses when provisioning DU single node clusters.

④ A block of IPv4 or IPv6 addresses in CIDR notation used for cluster services internal communication.

⑤ A plain text string. You can use the public key to SSH into the node after it has finished installing.

If you want to configure a static IP address for the managed cluster at this point, see the procedure in this document for configuring static IP addresses for managed clusters.

6. Create the `ClusterDeployment` custom resource:

```
apiVersion: hive.openshift.io/v1
kind: ClusterDeployment
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  baseDomain: <base_domain> ①
  clusterInstallRef:
    group: extensions.hive.openshift.io
    kind: AgentClusterInstall
    name: <cluster_name>
    version: v1beta1
  clusterName: <cluster_name>
  platform:
    agentBareMetal:
      agentSelector:
        matchLabels:
          cluster-name: <cluster_name>
  pullSecretRef:
    name: assisted-deployment-pull-secret
```

① The managed cluster's base domain.

7. Create the `KlusterletAddonConfig` custom resource:

```
apiVersion: agent.open-cluster-management.io/v1
kind: KlusterletAddonConfig
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterName: <cluster_name>
  clusterNamespace: <cluster_name>
  clusterLabels:
    cloud: auto-detect
    vendor: auto-detect
  applicationManager:
    enabled: true
  certPolicyController:
    enabled: false
  iamPolicyController:
    enabled: false
  policyController:
    enabled: true
  searchCollector:
    enabled: false ①
```

① Keep `searchCollector` disabled. Set to `true` to enable the `KlusterletAddonConfig` CR or `false` to disable the `KlusterletAddonConfig` CR.

8. Create the `ManagedCluster` custom resource:

```
apiVersion: cluster.open-cluster-management.io/v1
kind: ManagedCluster
metadata:
  name: <cluster_name>
spec:
  hubAcceptsClient: true
```

9. Create the `InfraEnv` custom resource:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  sshAuthorizedKey: <public_key> ①
  agentLabelSelector:
    matchLabels:
      cluster-name: <cluster_name>
  pullSecretRef:
    name: assisted-deployment-pull-secret
```

① Entered as plain text. You can use the public key to SSH into the target bare-metal host when it boots from the ISO.

10. Create the `BareMetalHost` custom resource:

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
  annotations:
    inspect.metal3.io: disabled
  labels:
    infraenvs.agent-install.openshift.io: "<cluster_name>"
spec:
  bootMode: "UEFI"
  bmc:
    address: <bmc_address> ①
    disableCertificateVerification: true
    credentialsName: <cluster_name>-bmc-secret
  bootMACAddress: <mac_address> ②
  automatedCleaningMode: disabled
  online: true
```

① The baseboard management console address of the installation ISO on the target bare-metal host.

② The MAC address of the target bare-metal host.

Optionally, you can add `bmac.agent-install.openshift.io/hostname: <host-name>` as an annotation to set the managed cluster's hostname. If you don't add the annotation, the hostname will default to either a hostname from the DHCP server or local host.

11. After you have created the custom resources, push the entire directory of generated custom resources to the Git repository you created for storing the custom resources.

Next steps

To provision additional clusters, repeat this procedure for each cluster.

Configuring BIOS for distributed unit bare-metal hosts

Distributed unit (DU) hosts require the BIOS to be configured before the host can be provisioned. The BIOS configuration is dependent on the specific hardware that runs your DUs and the particular requirements of your installation.

Procedure

1. Set the **UEFI/BIOS Boot Mode** to `UEFI`.
2. In the host boot sequence order, set **Hard drive first**.
3. Apply the specific BIOS configuration for your hardware. The following table describes a representative BIOS configuration for an Intel Xeon Skylake or Intel Cascade Lake server, based on the Intel FlexRAN 4G and 5G baseband PHY reference design.

The exact BIOS configuration depends on your specific hardware and network requirements. The following sample configuration is for illustrative purposes only.

Table 7. Sample BIOS configuration for an Intel Xeon Skylake or Cascade Lake server

BIOS Setting	Configuration
CPU Power and Performance Policy	Performance
Uncore Frequency Scaling	Disabled
Performance P-limit	Disabled
Enhanced Intel SpeedStep ® Tech	Enabled
Intel Configurable TDP	Enabled
Configurable TDP Level	Level 2
Intel® Turbo Boost Technology	Enabled
Energy Efficient Turbo	Disabled
Hardware P-States	Disabled
Package C-State	C0/C1 state
C1E	Disabled
Processor C6	Disabled

Enable global SR-IOV and VT-d settings in the BIOS for the host. These settings are relevant to bare-metal environments.

Configuring static IP addresses for managed clusters

Optionally, after creating the `AgentClusterInstall` custom resource, you can configure static IP addresses for the managed clusters.

You must create this custom resource before creating the `ClusterDeployment` custom resource.

Prerequisites

- Deploy and configure the `AgentClusterInstall` custom resource.

Procedure

1. Create a `NMStateConfig` custom resource:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: NMStateConfig
metadata:
 name: <cluster_name>
 namespace: <cluster_name>
 labels:
   sno-cluster-<cluster-name>: <cluster_name>
spec:
 config:
   interfaces:
     - name: eth0
       type: ethernet
       state: up
       ipv4:
         enabled: true
         address:
           - ip: <ip_address> ①
             prefix-length: <public_network_prefix> ②
         dhcp: false
   dns-resolver:
     config:
       server:
         - <dns_resolver> ③
   routes:
     config:
       - destination: 0.0.0.0/0
         next-hop-address: <gateway> ④
         next-hop-interface: eth0
         table-id: 254
 interfaces:
   - name: "eth0" ⑤
     macAddress: <mac_address> ⑥
```

① The static IP address of the target bare-metal host.

② The static IP address's subnet prefix for the target bare-metal host.

③ The DNS server for the target bare-metal host.

④ The gateway for the target bare-metal host.

⑤ Must match the name specified in the `interfaces` section.

⑥ The mac address of the interface.

2. When creating the `BareMetalHost` custom resource, ensure that one of its mac addresses matches a mac address in the `NMStateConfig` target bare-metal host.
3. When creating the `InfraEnv` custom resource, reference the label from the `NMStateConfig` custom resource in the `InfraEnv` custom resource:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  sshAuthorizedKey: <public_key>
  agentLabelSelector:
    matchLabels:
      cluster-name: <cluster_name>
  pullSecretRef:
    name: assisted-deployment-pull-secret
  nmStateConfigLabelSelector:
    matchLabels:
      sno-cluster-<cluster-name>: <cluster_name> # Match this label
```

Automated Discovery image ISO process for provisioning clusters

After you create the custom resources, the following actions happen automatically:

1. A Discovery image ISO file is generated and booted on the target machine.
2. When the ISO file successfully boots on the target machine it reports the hardware information of the target machine.
3. After all hosts are discovered, OpenShift Container Platform is installed.
4. When OpenShift Container Platform finishes installing, the hub installs the `klusterlet` service on the target cluster.
5. The requested add-on services are installed on the target cluster.

The Discovery image ISO process finishes when the `Agent` custom resource is created on the hub for the managed cluster.

Checking the managed cluster status

Ensure that cluster provisioning was successful by checking the cluster status.

Prerequisites

- All of the custom resources have been configured and provisioned, and the `Agent` custom resource is created on the hub for the managed cluster.

Procedure

1. Check the status of the managed cluster:

```
$ oc get managedcluster
```

`True` indicates the managed cluster is ready.

2. Check the agent status:

```
$ oc get agent -n <cluster_name>
```

3. Use the `describe` command to provide an in-depth description of the agent's condition. Statuses to be aware of include `BackendError`, `InputError`, `ValidationsFailing`, `InstallationFailed`, and `AgentIsConnected`. These statuses are relevant to the `Agent` and `AgentClusterInstall` custom resources.

```
$ oc describe agent -n <cluster_name>
```

4. Check the cluster provisioning status:

```
$ oc get agentclusterinstall -n <cluster_name>
```

5. Use the `describe` command to provide an in-depth description of the cluster provisioning status:

```
$ oc describe agentclusterinstall -n <cluster_name>
```

6. Check the status of the managed cluster's add-on services:

```
$ oc get managedclusteraddon -n <cluster_name>
```

7. Retrieve the authentication information of the `kubeconfig` file for the managed cluster:

```
$ oc get secret -n <cluster_name> <cluster_name>-admin-kubeconfig -o jsonpath
={.data.kubeconfig} | base64 -d > <directory>/<cluster_name>-kubeconfig
```

Configuring a managed cluster for a disconnected environment

After you have completed the preceding procedure, follow these steps to configure the managed cluster for a disconnected environment.

Prerequisites

- A disconnected installation of Red Hat Advanced Cluster Management (RHACM) 2.3.
- Host the `rootfs` and `iso` images on an HTTPD server.

Procedure

1. Create a `ConfigMap` containing the mirror registry config:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: assisted-installer-mirror-config
  namespace: assisted-installer
  labels:
    app: assisted-service
data:
  ca-bundle.crt: <certificate> ①
  registries.conf: |  ②
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

    [[registry]]
      location = <mirror_registry_url>  ③
      insecure = false
      mirror-by-digest-only = true
```

① The mirror registry's certificate used when creating the mirror registry.

② The configuration for the mirror registry.

③ The URL of the mirror registry.

This updates `mirrorRegistryRef` in the `AgentServiceConfig` custom resource, as shown below:

Example output

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
  namespace: assisted-installer
spec:
  databaseStorage:
    volumeName: <db_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <db_storage_size>
  filesystemStorage:
    volumeName: <fs_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <fs_storage_size>
  mirrorRegistryRef:
    name: 'assisted-installer-mirror-config'
  osImages:
    - openshiftVersion: <ocp_version>
      rootfs: <rootfs_url> ①
      url: <iso_url> ①
```

① Must match the URLs of the HTTPD server.

2. For disconnected installations, you must deploy an NTP clock that is reachable through the disconnected network. You can do this by configuring chrony to act as server, editing the `/etc/chrony.conf` file, and adding the following allowed IPv6 range:

```
# Allow NTP client access from local network.
#allow 192.168.0.0/16
local stratum 10
bindcmdaddress ::
allow 2620:52:0:1310::/64
```

Configuring IPv6 addresses for a disconnected environment

Optionally, when you are creating the `AgentClusterInstall` custom resource, you can configure IPV6 addresses for the managed clusters.

Procedure

1. In the `AgentClusterInstall` custom resource, modify the IP addresses in `clusterNetwork` and

`serviceNetwork` for IPv6 addresses:

```
apiVersion: extensions.hive.openshift.io/v1beta1
kind: AgentClusterInstall
metadata:
  # Only include the annotation if using OVN, otherwise omit the annotation
  annotations:
    agent-install.openshift.io/install-config-overrides:
'{"networking":{"networkType":"OVNKubernetes"}}'
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterDeploymentRef:
    name: <cluster_name>
  imageSetRef:
    name: <cluster_image_set>
  networking:
    clusterNetwork:
    - cidr: "fd01::/48"
      hostPrefix: 64
    machineNetwork:
    - cidr: <machine_network_cidr>
    serviceNetwork:
    - "fd02::/112"
  provisionRequirements:
    controlPlaneAgents: 1
    workerAgents: 0
  sshPublicKey: <public_key>
```

2. Update the `NMStateConfig` custom resource with the IPv6 addresses you defined.

Generating RAN policies

Prerequisites

- Install Kustomize
- Install the Kustomize Policy Generator plug-in

Procedure

1. Configure the `kustomization.yaml` file to reference the `policyGenerator.yaml` file. The following example shows the PolicyGenerator definition:

```
apiVersion: policyGenerator/v1
kind: PolicyGenerator
metadata:
  name: acm-policy
  namespace: acm-policy-generator
# The arguments should be given and defined as below with same order
--policyGenTempPath= --sourcePath= --outPath= --stdout --customResources
argsOneLiner: ./ranPolicyGenTempExamples ./sourcePolicies ./out true false
```

Where:

- `policyGenTempPath` is the path to the `policyGenTemp` files.
- `sourcePath`: is the path to the source policies.
- `outPath`: is the path to save the generated ACM policies.
- `stdout`: If `true`, prints the generated policies to the console.
- `customResources`: If `true` generates the CRs from the `sourcePolicies` files without ACM policies.

2. Test PolicyGen by running the following commands:

```
$ cd cnf-features-deploy/ztp/ztp-policy-generator/
```

```
$ XDG_CONFIG_HOME=./ kustomize build --enable-alpha-plugins
```

An `out` directory is created with the expected policies, as shown in this example:

```
out
├── common
│   ├── common-log-sub-ns-policy.yaml
│   ├── common-log-sub-oper-policy.yaml
│   ├── common-log-sub-policy.yaml
│   ├── common-pao-sub-catalog-policy.yaml
│   ├── common-pao-sub-ns-policy.yaml
│   ├── common-pao-sub-oper-policy.yaml
│   ├── common-pao-sub-policy.yaml
│   ├── common-policies-placementbinding.yaml
│   ├── common-policies-placementrule.yaml
│   ├── common-ptp-sub-ns-policy.yaml
│   ├── common-ptp-sub-oper-policy.yaml
│   ├── common-ptp-sub-policy.yaml
│   ├── common-sriov-sub-ns-policy.yaml
│   ├── common-sriov-sub-oper-policy.yaml
│   └── common-sriov-sub-policy.yaml
├── groups
│   ├── group-du
│   │   ├── group-du-mc-chronyd-policy.yaml
│   │   ├── group-du-mc-mount-ns-policy.yaml
│   │   ├── group-du-mcp-du-policy.yaml
│   │   ├── group-du-mc-sctp-policy.yaml
│   │   ├── group-du-policies-placementbinding.yaml
│   │   ├── group-du-policies-placementrule.yaml
│   │   ├── group-du-ptp-config-policy.yaml
│   │   └── group-du-sriov-operconfig-policy.yaml
│   └── group-sno-du
│       ├── group-du-sno-policies-placementbinding.yaml
│       ├── group-du-sno-policies-placementrule.yaml
│       ├── group-sno-du-console-policy.yaml
│       ├── group-sno-du-log-forwarder-policy.yaml
│       └── group-sno-du-log-policy.yaml
└── sites
    └── site-du-sno-1
        ├── site-du-sno-1-policies-placementbinding.yaml
        ├── site-du-sno-1-policies-placementrule.yaml
        ├── site-du-sno-1-sriov-nn-fh-policy.yaml
        ├── site-du-sno-1-sriov-nnp-mh-policy.yaml
        ├── site-du-sno-1-sriov-nw-fh-policy.yaml
        ├── site-du-sno-1-sriov-nw-mh-policy.yaml
        └── site-du-sno-1-.yaml
```

The common policies are flat because they will be applied to all clusters. However, the groups and sites have subdirectories for each group and site as they will be applied to different clusters.

Troubleshooting the managed cluster

Use this procedure to diagnose any installation issues that might occur with the managed clusters.

Procedure

1. Check the status of the managed cluster:

```
$ oc get managedcluster
```

Example output

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED   AVAILABLE   AGE
SNO-cluster     true                                  True     True        2d19h
```

If the status in the `AVAILABLE` column is `True`, the managed cluster is being managed by the hub.

If the status in the `AVAILABLE` column is `Unknown`, the managed cluster is not being managed by the hub. Use the following steps to continue checking to get more information.

2. Check the `AgentClusterInstall` install status:

```
$ oc get clusterdeployment -n <cluster_name>
```

Example output

```
NAME        PLATFORM           REGION   CLUSTERTYPE   INSTALLED    INFRAID
VERSION  POWERSTATE AGE
Sno0026    agent-baremetal                             false
Initialized
2d14h
```

If the status in the `INSTALLED` column is `false`, the installation was unsuccessful.

3. If the installation failed, enter the following command to review the status of the `AgentClusterInstall` resource:

```
$ oc describe agentclusterinstall -n <cluster_name> <cluster_name>
```

4. Resolve the errors and reset the cluster:

   a. Remove the cluster's managed cluster resource:

```
$ oc delete managedcluster <cluster_name>
```

b. Remove the cluster's namespace:

```
$ oc delete namespace <cluster_name>
```

This deletes all of the namespace-scoped custom resources created for this cluster. You must wait for the `ManagedCluster` CR deletion to complete before proceeding.

c. Recreate the custom resources for the managed cluster.

Updating managed policies with the Cluster Group Upgrades Operator

You can use the Cluster Group Upgrades Operator (CGUO) to manage the software lifecycle of multiple OpenShift clusters. CGUO uses Red Hat Advanced Cluster Management (RHACM) policies to perform changes on the target clusters.

The Cluster Group Upgrades Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview/.

Additional resources

- For more information about the Cluster Group Upgrades Operator (CGUO), see About the Cluster Group Upgrades Operator (CGUO).

About the auto-created ClusterGroupUpgrade CR for ZTP

The CGUO has a controller called `ManagedClusterForCGU` that monitors the `Ready` state of the `ManagedCluster` CRs on the hub cluster and creates the `ClusterGroupUpgrade` CRs for ZTP (zero touch provisioning).

For any managed cluster in the `Ready` state without a "ztp-done" label applied, the `ManagedClusterForCGU` controller automatically creates a `ClusterGroupUpgrade` CR in the `ztp-install` namespace with its associated RHACM policies that are created during the ZTP process. The CGUO then remediates the set of configuration policies that are listed in the auto-created `ClusterGroupUpgrade` CR to push the configuration CRs to the managed cluster.

If the managed cluster has no bound policies when the cluster becomes `Ready`, no `ClusterGroupUpgrade` CR is created.

Example of an auto-created `ClusterGroupUpgrade` CR for ZTP

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" ①
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" ②
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

① Applied to the managed cluster when the CGUO completes the cluster configuration.

② Applied to the managed cluster when the CGUO starts deploying the configuration policies.

End-to-end procedures for updating clusters in a disconnected environment

If you have deployed spoke clusters with distributed unit (DU) profiles using the GitOps ZTP with

the Cluster Group Upgrades Operator (CGUO) pipeline described in "Deploying distributed units at scale in a disconnected environment", this procedure describes how to upgrade your spoke clusters and Operators.

Preparing for the updates

If both the hub and the spoke clusters are running OpenShift Container Platform 4.9, you must update ZTP from version 4.9 to 4.10. If OpenShift Container Platform 4.10 is used, you can set up the environment.

Setting up the environment

The CGUO can perform both platform and Operator updates.

You must mirror both the platform image and Operator images that you want to update to in your mirror registry before you can use the CGUO to update your disconnected clusters. Complete the following steps to mirror the images:

- For platform updates, you must perform the following steps:
  1. Mirror the desired OpenShift Container Platform image repository. Ensure that the desired platform image is mirrored by following the "Mirroring the OpenShift Container Platform image repository" procedure linked in the Additional Resources. Save the contents of the `imageContentSources` section in the `imageContentSources.yaml` file:

     Example output

     ```
     imageContentSources:
      - mirrors:
        - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-
     dev/openshift4
        source: quay.io/openshift-release-dev/ocp-release
      - mirrors:
        - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-
     dev/openshift4
        source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
     ```

  2. Save the image signature of the desired platform image that was mirrored. You must add the image signature to the `PolicyGenTemplate` CR for platform updates. To get the image signature, perform the following steps:
     a. Specify the desired OpenShift Container Platform tag by running the following command:

        ```
        $ OCP_RELEASE_NUMBER=<release_version>
        ```

     b. Specify the architecture of the server by running the following command:

```
$ ARCHITECTURE=<server_architecture>
```

c. Get the release image digest from Quay by running the following command

```
$ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-
release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From:
.*@//p')"
```

d. Set the digest algorithm by running the following command:

```
$ DIGEST_ALGO="${DIGEST%%:*}"
```

e. Set the digest signature by running the following command:

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```

f. Get the image signature from the mirror.openshift.com website by running the following command:

```
$ SIGNATURE_BASE64=$(curl -s
"https://mirror.openshift.com/pub/openshift-
v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signatu
re-1" | base64 -w0 && echo)
```

g. Save the image signature to the `checksum-<OCP_RELEASE_NUMBER>.yaml` file by running the following commands:

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. Prepare the update graph. You have two options to prepare the update graph:

a. Use the OpenShift Update Service.

For more information about how to set up the graph on the hub cluster, see Deploy the operator for OpenShift Update Service and Build the graph data init container.

b. Make a local copy of the upstream graph. Host the update graph on an `http` or `https` server in the disconnected environment that has access to the spoke cluster. To download the update graph, use the following command:

```
$ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel
=stable-4.10 -o ~/upgrade-graph_stable-4.10
```

- For Operator updates, you must perform the following task:
  - Mirror the Operator catalogs. Ensure that the desired operator images are mirrored by following the procedure in the "Mirroring Operator catalogs for use with disconnected clusters" section.

Additional resources

- For more information about how to update ZTP, see Upgrading GitOps ZTP.
- For more information about how to mirror an OpenShift Container Platform image repository, see Mirroring the OpenShift Container Platform image repository.
- For more information about how to mirror Operator catalogs for disconnected clusters, see Mirroring Operator catalogs for use with disconnected clusters.
- For more information about how to prepare the disconnected environment and mirroring the desired image repository, see Preparing the disconnected environment.
- For more information about update channels and releases, see Understanding upgrade channels and releases.

Performing a platform update

You can perform a platform update with the CGUO.

Prerequisites

- Install the Cluster Group Upgrades Operator (CGUO).
- Update ZTP to the latest version.
- Provision one or more managed clusters with ZTP.
- Mirror the desired image repository.
- Log in as a user with `cluster-admin` privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Create a `PolicyGenTemplate` CR for the platform update:
   a. Save the following contents of the `PolicyGenTemplate` CR in the `du-upgrade.yaml` file.

   Example of `PolicyGenTemplate` for platform update

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
```

```
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: ImageSignature.yaml ①
      policyName: "platform-upgrade-prep"
      binaryData:
        ${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64} ②
    - fileName: DisconnectedICSP.yaml
      policyName: "platform-upgrade-prep"
      metadata:
        name: disconnected-internal-icsp-for-ocp
      spec:
        repositoryDigestMirrors: ③
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
            source: quay.io/openshift-release-dev/ocp-release
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
            source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
    - fileName: ClusterVersion.yaml ④
      policyName: "platform-upgrade-prep"
      metadata:
        name: version
        annotations:
          ran.openshift.io/ztp-deploy-wave: "1"
      spec:
        channel: "stable-4.10"
        upstream: http://upgrade.example.com/images/upgrade-graph_stable-
4.10
    - fileName: ClusterVersion.yaml ⑤
      policyName: "platform-upgrade"
      metadata:
        name: version
      spec:
        channel: "stable-4.10"
        upstream: http://upgrade.example.com/images/upgrade-graph_stable-
4.10
        desiredUpdate:
          version: 4.10.4
      status:
        history:
          - version: 4.10.4
            state: "Completed"
```

① The `ConfigMap` CR contains the signature of the desired release image to update to.

② Shows the image signature of the desired OpenShift Container Platform release. Get the signature from the `checksum-${OCP_RELASE_NUMBER}.yaml` file you saved when following the procedures in the "Setting up the environment" section.

③ Shows the mirror repository that contains the desired OpenShift Container Platform image. Get the mirrors from the `imageContentSources.yaml` file that you saved when following the procedures in the "Setting up the environment" section.

④ Shows the `ClusterVersion` CR to update upstream.

⑤ Shows the `ClusterVersion` CR to trigger the update. The `channel`, `upstream`, and `desiredVersion` fields are all required for image pre-caching.

The `PolicyGenTemplate` CR generates two policies:

- The `du-upgrade-platform-upgrade-prep` policy does the preparation work for the platform update. It creates the `ConfigMap` CR for the desired release image signature, creates the image content source of the mirrored release image repository, and updates the cluster version with the desired update channel and the update graph reachable by the spoke cluster in the disconnected environment.
- The `du-upgrade-platform-upgrade` policy is used to perform platform upgrade.

b. Add the `du-upgrade.yaml` file contents to the `kustomization.yaml` file located in the ZTP Git repository for the `PolicyGenTemplate` CRs and push the changes to the Git repository.

ArgoCD pulls the changes from the Git repository and generates the policies on the hub cluster.

c. Check the created policies by running the following command:

```
$ oc get policies -A | grep platform-upgrade
```

2. Apply the required update resources before starting the platform update with the CGUO.

a. Save the content of the `platform-upgrade-prep ClusterUpgradeGroup` CR with the `du-upgrade-platform-upgrade-prep` policy and the target spoke clusters to the `cgu-platform-upgrade-prep.yml` file, as shown in the following example:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: true
```

b. Apply the policy to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-upgrade-prep.yml
```

c. Monitor the update process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```

3. Create the `ClusterGroupUpdate` CR for the platform update with the `spec.enable` field set to `false`.

a. Save the content of the platform update `ClusterGroupUpdate` CR with the `du-upgrade-platform-upgrade` policy and the target clusters to the `cgu-platform-upgrade.yml` file, as shown in the following example:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade
    preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

b. Apply the `ClusterGroupUpdate` CR to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-upgrade.yml
```

4. Optional: Pre-cache the images for the platform update.

a. Enable pre-caching in the `ClusterGroupUpdate` CR by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
platform-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. Monitor the update process and wait for the pre-caching to complete. Check the status of pre-caching by running the following command on the hub cluster:

```
$ oc get cgu cgu-platform-upgrade -o jsonpath='{.status.precaching.status}'
```

5. Start the platform update:

   a. Enable the `cgu-platform-upgrade` policy and disable pre-caching by running the following command:

   ```
   $ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
   platform-upgrade \
   --patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
   ```

   b. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

   ```
   $ oc get policies --all-namespaces
   ```

Additional resources

- For more information about mirroring the images in a disconnected environment, Preparing the disconnected environment

Performing an Operator update

You can perform an Operator update with the CGUO.

Prerequisites

- Install the Cluster Group Upgrades Operator (CGUO).
- Update ZTP to the latest version.
- Provision one or more managed clusters with ZTP.
- Mirror the desired index image, bundle images, and all Operator images referenced in the bundle images.
- Log in as a user with `cluster-admin` privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Update the `PolicyGenTemplate` CR for the Operator update.

   a. Update the `du-upgrade PolicyGenTemplate` CR with the following additional contents in the `du-upgrade.yaml` file:

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "operator-catsrc-policy"
      metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v4.10 ①
        updateStrategy: ②
          registryPoll:
            interval: 10m
```

① The index image URL contains the desired Operator images. If the index images are always pushed to the same image name and tag, this change is not needed.

② Set how frequently Operator Lifecycle Manager (OLM) pulls the index image for new Operator versions with the `registrypoll.interval` field. This change is not needed if a new index image tag is always pushed for y-stream and z-stream Operator updates.

b. This update generates one policy, `du-upgrade-operator-catsrc-policy`, to update the `redhat-operators` catalog source with the new index images that contain the desired Operators images.

If you want to use the image pre-caching for Operators and there are Operators from a different catalog source other than `redhat-operators`, you must perform the following tasks:

- Prepare a separate catalog source policy with the new index image or registry poll interval update for the different catalog source.
- Prepare a separate subscription policy for the desired Operators that are from the different catalog source.

For example, the desired SRIOV-FEC Operator is available in the `certified-operators` catalog source. To update the catalog source and the Operator subscription, add the following contents to generate two policies, `du-upgrade-fec-catsrc-policy` and `du-upgrade-subscriptions-fec-policy`:

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
       ...
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "fec-catsrc-policy"
      metadata:
        name: certified-operators
      spec:
        displayName: Intel SRIOV-FEC Operator
        image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
        updateStrategy:
          registryPoll:
            interval: 10m
    - fileName: AcceleratorsSubscription.yaml
      policyName: "subscriptions-fec-policy"
      spec:
        channel: "stable"
        source: certified-operators
```

c. Remove the specified subscriptions channels in the common `PolicyGenTemplate` CR, if they exist. The default subscriptions channels from the ZTP image are used for the update.

The default channel for the Operators applied through ZTP 4.10 is `stable`, except for the `performance-addon-operator`. The default channel for PAO is `4.10`. You can also specify the default channels in the common `PolicyGenTemplate` CR.

d. Push the `PolicyGenTemplate` CRs updates to the ZTP Git repository.

ArgoCD pulls the changes from the Git repository and generates the policies on the hub cluster.

e. Check the created policies by running the following command:

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. Apply the required catalog source updates before starting the Operator update.

a. Save the content of the `ClusterGroupUpgrade` CR named `operator-upgrade-prep` with the catalog source policies and the target spoke clusters to the `cgu-operator-upgrade-prep.yml` file:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1
```

b. Apply the policy to the hub cluster by running the following command:

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

c. Monitor the update process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. Create the `ClusterGroupUpgrade` CR for the Operator update with the `spec.enable` field set to `false`.

a. Save the content of the Operator update `ClusterGroupUpgrade` CR with the `du-upgrade-operator-catsrc-policy` policy and the subscription policies created from the common `PolicyGenTemplate` and the target clusters to the `cgu-operator-upgrade.yml` file, as shown in the following example:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-operator-catsrc-policy ①
  - common-subscriptions-policy ②
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

① The policy is needed by the image pre-caching feature to retrieve the operator images from the catalog source.

② The policy contains Operator subscriptions. If you have upgraded ZTP from 4.9 to 4.10 by following "Upgrade ZTP from 4.9 to 4.10", all Operator subscriptions are grouped into the `common-subscriptions-policy` policy.

One `ClusterGroupUpgrade` CR can only pre-cache the images of the desired Operators defined in the subscription policy from one catalog source included in the `ClusterGroupUpgrade` CR. If the desired Operators are from different catalog sources, such as in the example of the SRIOV-FEC Operator, another `ClusterGroupUpgrade` CR must be created with `du-upgrade-fec-catsrc-policy` and `du-upgrade-subscriptions-fec-policy` policies for the SRIOV-FEC Operator images pre-caching and update.

b. Apply the `ClusterGroupUpgrade` CR to the hub cluster by running the following command:

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. Optional: Pre-cache the images for the Operator update.

a. Before starting image pre-caching, verify the subscription policy is `NonCompliant` at this point by running the following command:

```
$ oc get policy common-subscriptions-policy -n <policy_namespace>
```

Example output

```
NAME                          REMEDIATION ACTION   COMPLIANCE STATE     AGE
common-subscriptions-policy   inform               NonCompliant         27d
```

b. Enable pre-caching in the `ClusterGroupUpgrade` CR by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
operator-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

c. Monitor the process and wait for the pre-caching to complete. Check the status of pre-caching by running the following command on the spoke cluster:

```
$ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
```

d. Check if the pre-caching is completed before starting the update by running the following command:

```
$ oc get cgu -n default cgu-operator-upgrade -ojsonpath
='{.status.conditions}' | jq
```

Example output

```
[
    {
      "lastTransitionTime": "2022-03-08T20:49:08.000Z",
      "message": "The ClusterGroupUpgrade CR is not enabled",
      "reason": "UpgradeNotStarted",
      "status": "False",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-03-08T20:55:30.000Z",
      "message": "Precaching is completed",
      "reason": "PrecachingCompleted",
      "status": "True",
      "type": "PrecachingDone"
    }
]
```

5. Start the Operator update.

a. Enable the `cgu-operator-upgrade` `ClusterGroupUpgrade` CR and disable pre-caching to start the Operator update by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
operator-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. Monitor the process. Upon completion, ensure that the policy is compliant by running

the following command:

```
$ oc get policies --all-namespaces
```

Additional resources

- For more information about updating GitOps ZTP, see Upgrading GitOps ZTP.

Performing a platform and an Operator update together

You can perform a platform and an Operator update at the same time.

Prerequisites

- Install the Cluster Group Upgrades Operator (CGUO).
- Update ZTP to the latest version.
- Provision one or more managed clusters with ZTP.
- Log in as a user with `cluster-admin` privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Create the `PolicyGenTemplate` CR for the updates by following the steps described in the "Performing a platform update" and "Performing an Operator update" sections.
2. Apply the prep work for the platform and the Operator update.
   a. Save the content of the `ClusterGroupUpgrade` CR with the policies for platform update preparation work, catalog source updates, and target clusters to the `cgu-platform-operator-upgrade-prep.yml` file, for example:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-operator-catsrc-policy
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

   b. Apply the `cgu-platform-operator-upgrade-prep.yml` file to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

c. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```

3. Create the `ClusterGroupUpdate` CR for the platform and the Operator update with the `spec.enable` field set to `false`.

a. Save the contents of the platform and Operator update `ClusterGroupUpdate` CR with the policies and the target clusters to the `cgu-platform-operator-upgrade.yml` file, as shown in the following example:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade ①
  - du-upgrade-operator-catsrc-policy ②
  - common-subscriptions-policy ③
  preCaching: true
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

① This is the platform update policy.

② This is the policy containing the catalog source information for the Operators to be updated. It is needed for the pre-caching feature to determine which Operator images to download to the spoke cluster.

③ This is the policy to update the Operators.

b. Apply the `cgu-platform-operator-upgrade.yml` file to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. Optional: Pre-cache the images for the platform and the Operator update.

a. Enable pre-caching in the `ClusterGroupUpgrade` CR by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. Monitor the update process and wait for the pre-caching to complete. Check the status of pre-caching by running the following command on the spoke cluster:

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

c. Check if the pre-caching is completed before starting the update by running the following command:

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

5. Start the platform and Operator update.

a. Enable the `cgu-du-upgrade` `ClusterGroupUpgrade` CR to start the platform and the Operator update by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```

The CRs for the platform and Operator updates can be created from the beginning by configuring the setting to `spec.enable: true`. In this case, the update starts immediately after pre-caching completes and there is no need to manually enable the CR.

Both pre-caching and the update create extra resources, such as policies, placement bindings, placement rules, managed cluster actions, and managed cluster view, to help complete the procedures. Setting the `afterCompletion.deleteObjects` field to `true` deletes all these resources after the updates complete.