Edge-Node Clustering Design

Introduction

The document has the Edge-node Cluster design considerations and detailed procedures for Cluster Creation; Network Instance Deployment; Volume Instance Deployment; Application Deployment; Application Migration and Status for both Controller and Onsite edge-nodes. It also has the Edge-Node API for the clustering operation.

EVE development has the extension to build the kubevirt image with kubernetes and kubevirt managing the user VMs, see document Cluster compute and storage support in EVE. This Edge-Node Clustering is built on top of that development.

Terminology

- Cluster - Edge-Node Cluster, with Kubernetes cluster on EVE edge-nodes, at least 3 local network connected nodes, managing the User applications on EVE devices at customer sites.
- Master - Kubernetes master node or server node, which runs with control and worker functions. In the initial phases, we only support exactly 3 master nodes for a cluster.
- Cluster-Interface - The EVE device physical Interface which the cluster uses for kubernetes communication among the kubernetes nodes. This may or may not be part of the EVE device management ports. It can be a Kubernetes dedicated ‘App Shared’ port on the device.
- Cluster-Prefix - A dedicated IP prefix is used on the ‘Cluster-Interface’ for kubernetes cluster to communicate with. This is configured regardless of the DHCP or manual usage on the site. It can be the secondary IP subnet on the interface.
- Seed-server - One of the master nodes is designated by the configuration to be the ‘Seed-server’ in the cluster. The ‘seed-node’ is responsible for generating the initial kubernetes configuration, the certificates, and the token in the cluster for other master nodes to join. A ‘seed-node’ may be changed from one device to another if the original ‘seed-node’ can not be brought up again.
- SSIP - Seed-Server IP address. This is used during initial bootstrap the cluster for the Non-Seed-Node to join the cluster.
- PVC - Persistent Volume Claim. User application and its volumes handle which is managed from kubernetes and longhorn systems.
- Kubevirt Image - EVE image compiled with ‘kubevirt’ HV mode, instead of the currently offered ‘kvm’ and ‘xen’ modes. All the cluster devices need to run this type of image.
- Single Node - A single kubernetes edge-node running with EVE Kubevirt image, the edge-node launches VM and container using kubernetes mechanism. This document does not discuss the details of single-node. If a cluster loses two edge-nodes on-site, then the only live edge-node can reduce to the single-node mode.
- Kube Container - An EVE service which is a `services` container, running in system containerd, parallel to wwan and pillar, called ‘kube’, which is responsible for kubernetes cluster either on Edge-Node Cluster or on a single-node. It runs the K3S, Longhorn, KubeVirt, etc. and EVE specific changed open-source kubernetes software.
- HA - High Availability for Cluster. In particular this is high availability for the kubernetes control plane functions. For the 3 nodes cluster, it can have one node down or disconnected but the cluster will still be functional for the kubernetes operations.
- ResourceLabel - Node or edge-node resource label. It is a pair of key and value strings, to represent some particular resource on an edge-node. This is used for Apps, NIs and VIs placement and App placement and migration be restricted on certain edge-nodes
- Auto-Migration - An user application can be automatically migrated from original node onto any node which has resources.
- Manual-Migration - An user application can only be migrated from the original node onto another node by user’s configuration change through the controller.
- Server Node - Kubernetes server or master node. We initially only support the cluster with 3 server nodes. The server nodes run control plane processes such as API-Server, Scheduler, Controller-Manager, Etcd, etc. They also serve work loads for running VMs and containers in our case. Unlike the ‘Server’ nodes, the agent nodes don’t have most of the kubernetes controller functions and they only run work loads.
- Agent Node - Kubernetes agent node or worker node. It does not run most of the processes with controller plane functionalities of the kubernetes cluster. Those agent nodes can not be one of the 3 HA controller kubernetes nodes.
- NI - Network Instance of edge-nodes
- VI - Volume Instance of edge-nodes
- DNId - Designated Node ID for a cluster application. The UUID of the edge-node in the cluster responsible for launching the application.

Cluster Workflow

We first go into the workflow of edge-node clustering. 1) Create a cluster; 2) Create Network Instance(s) and Volume Instances if any 3) Deploy an Application on the edge-node cluster. 4) Move an application manually or automatically among cluster nodes 5) Seed-Server change on cluster 6) Adding new node to existing cluster 7) Delete a node from the cluster 8) Replace a node in the cluster

Create Edge-Node Cluster

There are two steps for the cluster creation process, the first step is to create it in the Controller, and the second step is on site the devices need to form a kubernetes cluster among themselves upon receiving the configuration from the Controller.

Controller Edge-Node Cluster Creation

1. Assign unique cluster name and UUID
2. Select 3 edge-nodes, by some mechanism
3. Designate an edge-node as ‘seed-server’, either manually or automatically

4. For each edge-node, designate one physical port as the ‘Cluster-Interface’. This port can be either a Mgmt type or it can be ‘App Shared’ and it should be consistent in the cluster.
5. Cluster can define ‘ResourceLabels’ if needed. Each edge-node can be attached with zero or multiple ResourceLabels. For instance, ResourceLabels can be ‘GPU:2’ or ‘DirectConnect:eth3’. Those can be used when a user application needs to be satisfied with certain hardware resources during Kubernetes dynamic scheduling. If the cluster comprises identical devices, then there is no need to do the ResourceLabel for it.
6. Generate a bearer token for the cluster, with 32 bytes of length, see the k3s doc. This bearer token is created by the controller, and is passed to the seed node to use for future nodes that join the cluster, as well as other nodes to use to join the cluster. Needs to be encrypted with device’s public certificate when downloading with the device configuration
7. Add to above #4, a ‘Cluster-Interface’ needs to be selected with a real physical port name.
8. Regardless of of above ‘Cluster-Interface’ is using DHCP or Static IP assignment, we may always create a new IP prefix on that ‘Cluster-Interface’, let's call this ‘Cluster-Prefix’, we can automatically assign this e.g. ‘10.244.244.0/28’ to allow say 16 devices to form a cluster on site. But we should give users a choice of defining their own if there is a conflict on this prefix or some other reasons. The controller will automatically assign the IP address to each of the edge-node within this ‘Cluster-Prefix’ range.
9. Controller is responsible for creating the cluster with the set of device configurations for each of the edge-nodes in the cluster, but forming the cluster on site depends on collective edge-node’s bootstrap processing of the EVE pillar and kubernetes software.
10. We may need to add an edge-node to the cluster later, either as a server in the case of a 5 server HA cluster or a replacement edge-node for an original edge-node; or as an agent node for the cluster.
11. Controller deployment policy may be used for some of the above cluster wide configurations.

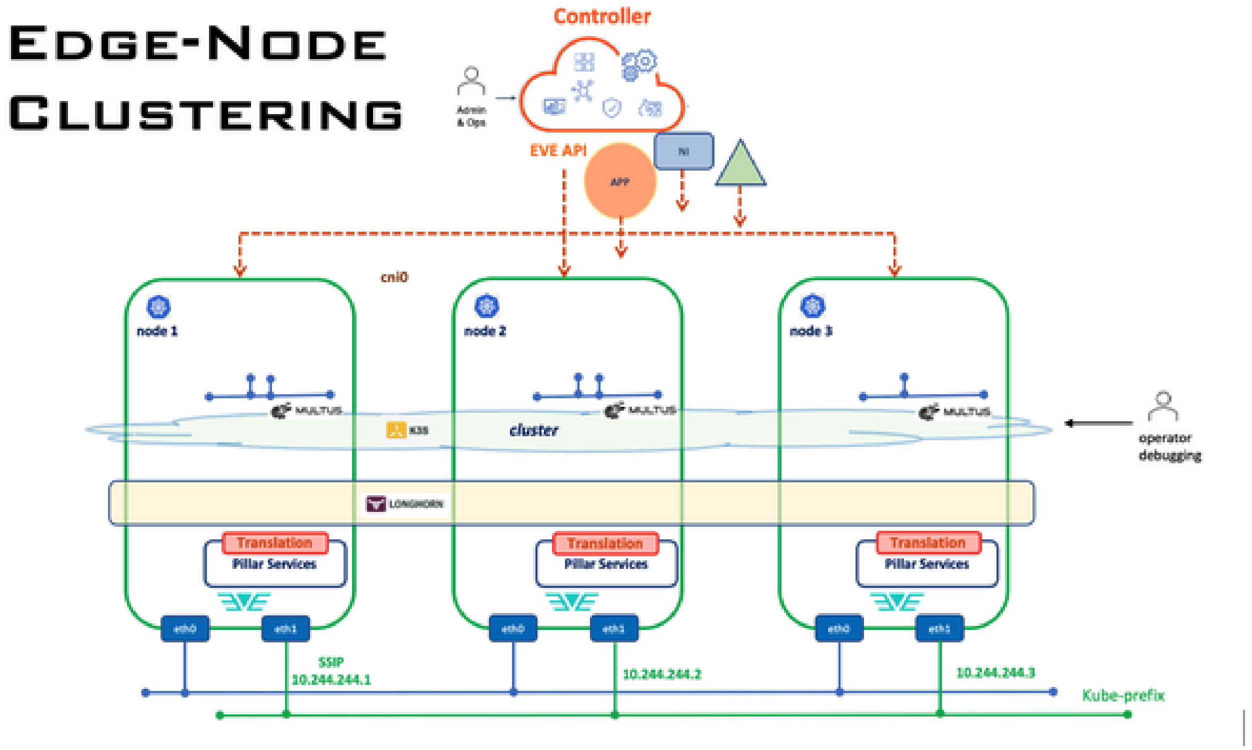
On-Site Edge-Node Cluster Bootstrap

- Each edge-node gets its own configuration from Controller, including the EdgeNodeCluster API.
- We require all the edge-nodes in the cluster share the same local network on site.
- The pillar side needs to configure the ‘Cluster-Prefix’ on the ‘Cluster-Interface’ to be used for kubernetes on each edge-node.
- The bootstrap process in kubernetes depends on if the edge-node is a ‘Seed-Server’ or not.
    - Is Seed-Server: It, the bootstrap functions in pillar, cluster-init.sh, etc, uses the ‘Cluster-Prefix’, cluster token and the Cluster-Interface to start the ‘k3s server’ with ‘cluster-init’. During the kubernetes cluster bring up, the SSIP is configured. the ‘Kube-VIP’ will use this VIP address and provision the VIP onto the Cluster-Interface port. Thus, this interface will have two IP addresses. One is the Interface IP address, the other is the Virtual IP address. They are in the same IP subnet. The kubernetes also builds the server certificates to include this VIP as one of the SANs.
    - Is Non Seed-Server: the cluster-init.sh of the kubernetes is monitoring this IP status, it will only start to provision the kubernetes if the ‘Seed-Server’ IP address status is pingable. As long as the ‘Seed-Server’ IP (SSIP) is alive, the cluster-init.sh continues with the kubernetes bring-up to use the SSIP as the cluster IP address to join and the token as the cluster secret.
    - The Seed-Server and Non Seed-Server uses different ‘k3s server’ configure files, we call one as seed-config.yaml and the other config.yaml. Only the Seed-Server may use the ‘seed-config.yaml’.
- There is no particular sequence the edge-nodes need to be provisioned from the Controller, for Seed-Server or Non Seed-Server nodes. They can be onboarded and provisioned with Edge-Node Clustering in the factory and be brought to the site to form the kubernetes cluster, with or without the Controller.
- If the role of the node is an ‘Agent', it uses the same bootstrap processing as mentioned in the above server node, except for it starts the ‘k3s agent’ command with the ‘agent_config.yaml’.
- Apply the ‘ResourceLabels’ download to the kubernetes node object of this edge-node and handle the label changes dynamically
- ‘EdgeNodeCluster’ API is described in the ‘Edge-Node Cluster APIs Changes’ section

On-Site Edge-Node Networking in Clustering

1. A. For the Cluster-Interface and Cluster-Prefix, it will have an extra IP prefix to be configured on the device port. Service ‘nim’ probably is the right place to perform this.
2. For the new Cluster-Prefix, the IP rules and ACLs need to be evaluated and installed in addition to the current single-node mode.
3. In addition to the single-node case, this cluster is a kubernetes distributed system and different modules communicate with each other using different TCP ports. We need to open more to allow inbound kubernetes service IP packets(the options are, if we just open those ports, or we implement some checks on allowing packets to those ports with certain source IP address being on the same subnet):
    a. 53 - DNS, or CoreDNS port, this is there in single-node also
    b. 6443 - basic cluster api-server endpoint port
    c. 2379-2381 - Etcd endpoints ports
    d. 8472 - Flannel overlay port
    e. 9500-9503 - Longhorn system ports
    f. 8000-8002 - Longhorn Webhook ports
    g. 3260 - Iscsi port
    h. 2049 - NFS port
4. In kubevirt single-node case, one of the IP rules for port is added with ‘cni0’ subnet route to allow kubernetes internal traffic. In cluster mode, this change needs to be modified to handle inter-device kubernetes traffic.

Cluster Creation Diagram For Devices



Network Instance (NI) Deployment in Cluster

Controller NI Provisioning

1. New cluster wide NI vs current edge-node NI
2. Same NI name and UUID be configured onto each of the edge-node in the cluster
3. For ‘Local’ type of ‘Auto’ or ‘Manual’ IP Configuration, We may need to add a choice of ‘Same Prefix’ or ‘Different Prefix’ for NI on each of the edge-nodes in the cluster. The ‘Different Prefix’ scheme may be useful in the future if we want to allow one App in edge-node ‘A’ to communicate with another App in edge-node ‘B’. Some CNI such as ‘Calico’ allow multiple IP prefixes on the same kubernetes node to be redistributed in the cluster. For ‘Different Prefix’ mode, Controller assigns e.g. 10.1.10.0/24 to first edge-node for the NI (with the same name/UUID), 10.1.11/0/24 to the 2nd edge-node, and 10.1.12.0/24 to the 3rd node of the NI.
4. May want to let users pick which edge-nodes need this NI deployment, or the deployment can be bound by the ‘ResourceLabels’ of the edge-nodes. If an App can only run on the edge-node with ResourceLabel of ‘foo’, then the NI used by this App makes sense to be only deployed onto the same set of edge-nodes. In particular, in the future, we may have more than 3 nodes in the cluster.
5. The ResourceLabel list of the NI does not need to be downloaded onto the edge-node, it can be used for the controller side to filter out certain edge-nodes. Thus there is no new addition to the NetworkInstanceConfig API.
6. Once each edge-node NI configuration is built, it will be used in the device configuration for each edge-node selected
7. Should we reuse the Library’s ‘Network Instance’ object, probably?
8. Should we have default NI for a cluster? Probably yes just as in the single-node case. Should be default to ‘Same Prefix’ or to ‘Different Prefix’ for the ‘default’ NI?
9. Normally once a NI is configured and it is being downloaded onto the edge-node. We may want to hold this in the controller until the App is attached to it. In the ‘Manual-Migration’ case, only the DNId edge-node will get this NI, while in ‘Auto-Migration’ case, all the edge-nodes get this NI configuration. Multiple App-Instance may use the same NI, so as long as at least one App is using it, it needs to be downloaded. The NI does not need to have ‘DNId’ string attached (unlike the VI case which consumes valuable resources)
10. Normally a NI specifies one port or ‘uplink’ to external connection, in the cluster case, a ‘port’ can be on different devices. We need to let users pick device:port pairs for all the devices (3 in this case) involved in Application using the NI.

On-Site Edge-Node NI Handling

- It does not require special handling of Network Instance for the cluster
- No API change needed for the NI configuration to edge-nodes

Volume Instance (VI) Deployment in Cluster

Controller Volume Instance Provisioning

1. Need to extend the type ‘filesystem storage’, currently only allows ‘Block Storage’ for non image usage.
2. Same as in the NI case, the same VI name and UUID can be deployed onto each of the edge-nodes in the cluster.
3. Same as in the NI case, may want to let users pick which edge-nodes need this VI or assign the ResourceLabel requirement.
4. Same as in the NI case, if the ResourceLabel list exists, it does not need to be downloaded to the edge-nodes.
5. Once each edge-node VI configuration is built, it will be used in the device configuration for each edge-node selected
6. If the VI does not need to be replicated by the cluster, then the user should only pick one edge-node
7. Same as above NI question, should we reuse the same ‘VI’ object or create a new one for Edge-node Cluster VI?
8. Normally once a VI is configured independent of the Application, it is downloaded onto the edge-node. This cluster case, we should allow users to pick up to three devices for this VI in the cluster. When the App is attached with this VI and if the App device selection does not match the VI device selection, we generate an error and ask user to modify. In the ‘Manual Migration’ case, only the DNId will get this VI configuration. In the ‘Auto-Migration’ of the App case, the other nodes may also get the VI configuration. A new DNId string is added to the ‘Volume’ API.

On-Site Edge-Node VI Handling

- For each VI provisioned on the edge-node, need to create the volume on the device
- Need to watch the Volume.HasNoAppReference status, if is true, do not create PVC on the device, since only one edge-node in the cluster should create PVC for the kubernetes, and there is only one storage block which is shared among all the edge-nodes. Volume can still be created by the ‘containerd’.
- The Volume and PVC will be created when the VolumeConfig DNId string is the edge-node UUID. If the DNId does not match, then the code needs to be modified on the EVE pillar side to handle the volume instance configuration, but not to create volume and PVC.
- The API change, is to add the DNId string, needed for the VI configuration

Cluster Application Deployment

Controller Application Deployment

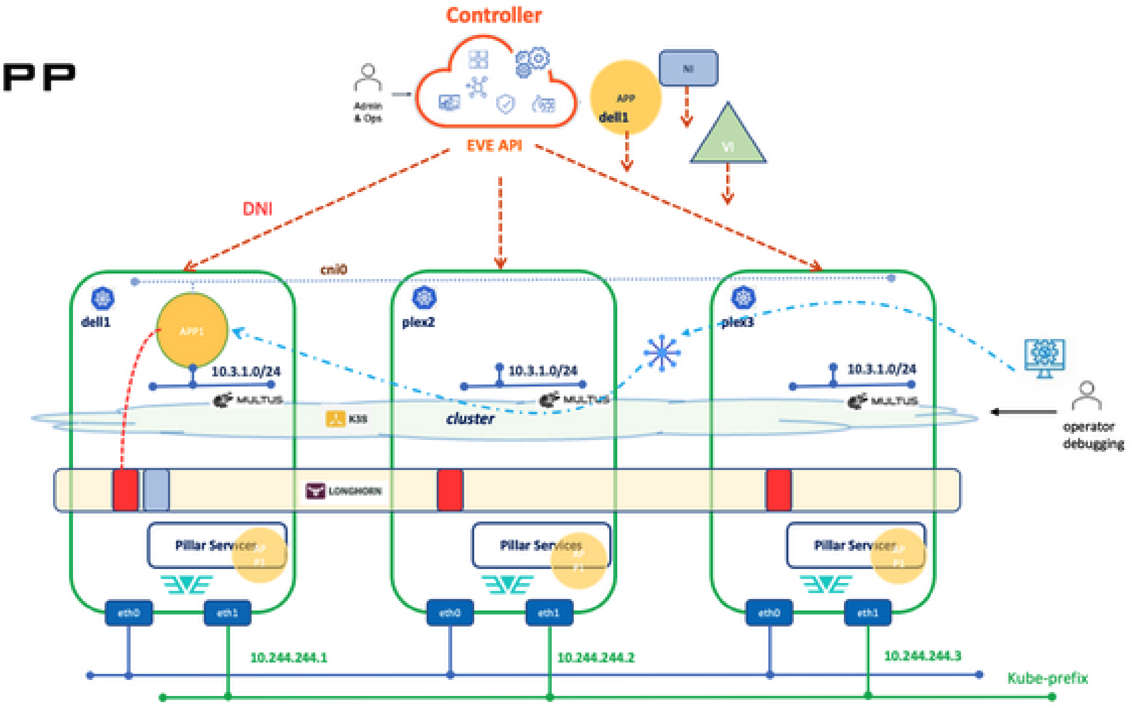
1. An Application (App-Bundle or App manifest) is being deployed on a cluster, rather than on a device
2. The ResourceLabels may not be implemented in the first phases of the clustering.
3. Users can manually pick an edge-node in the cluster for the initial placement, if the user does not pick, then the system automatically assigns one, in particular based on the requirement of the ResourceLabels needed for the App. The picking by users is optional, just like the ResourceLables is optional. System may need to automatically assign the App to a node. Users may not know which device to pick, but they may know which resources this App is needed.
4. Users should specify if this placement is allowed to be Auto-Migrated onto a different edge-node in the cluster or not, by user specify, it means the controller needs to decide if the App’s resource requirement matches the nodes, or pinned to the initial node.
5. If Auto-Migration is allowed, then users can pick a list of ‘ResourceLabels’ for the requirement of the migration, otherwise, it can be migrated onto any edge-node which has resources in the case the initial edge-node is down. If the App has ResourceLabels configured, maybe the controller will only show the eligible nodes for migration purposes. Since ResourceLabels is not in the first phases, users have no ResourceLabels to pick.
6. The UUID of the Designated Node for App (DNId) is set for the edge-node which is responsible for launching the application on the edge-node through kubernetes. For instance, when the initial placement is set for edge-node ‘A’ for this App Instance, the DNId is included in the AppInstanceConfig API when downloading the configuration to edge-node ‘A’.
7. DNId is added to the AppInstanceConfig API, see section ‘Cluster API Changes’
8. ‘AutoMigration’ and ‘NodeSelectLabels’ (not in first phases) also need to be downloaded to edge-nodes

On-Site App Instance Deployment

- If the edge-node is a Server or Agent (defined in EdgeNodeCluster API), when it receives the AppInstanceConfig from the Controller, it sets the AppInstanceConfig.Activate to ‘false’ unless the ‘DNId’ matches on this edge-node.
- Similarly set the Volume.HasNoAppReference to ‘true’ in the VolumeConfig unless the ‘DNId’ matches.
- For the edge-node which the App DNId matches, it handles the application similar to the case on the single-node case. plus:
    - If the App for this edge-node has ‘AutoMigration’ is ‘true’, the ‘ReplicaSet’ of this App is used and uses the ‘PreferredDuringScheduling’ to this edge-node. If the ‘NodeSelectLabels’ is not empty in the API, include this to the NodeSelector (not in first phases) for scheduling configuration.
    - If the App for this edge-node has ‘AutoMigration’ is ‘false’, the ‘Pod’ of the App is used for ‘NodeSelector’ to this edge-node

Cluster Application Deployment Diagram

APP



Cluster Application Migrate (Manual)

Controller Application Manual Migrate

1. Manual migration can be applied to the Apps with ‘AutoMigration’ set or unset.
2. Users can decide to migrate the Application from one edge-node to another. Similar to the initial placement of the App, users may manually pick one edge-node, or let the system pick automatically based on ResourceLabels of the App.
3. The ‘DNId’ string is changed from the previous edge-node to the newly selected edge-node
4. Download the modified configuration to all the edge-nodes in the cluster

On-Site Application Manual Migrate

- AppInstanceConfig Active will be set on the new edge-node, which triggers the pillar normal App handling and processing.
- The new edge-node may need to remove the original Pod or ReplicaSet from the kubernetes first, then recreate the new one with changed NodeSelector requirements
- The previous edge-node may be unreachable. When it is online, the AppInstanceConfig Active flag is reset on the previous edge-node. Make sure it does not try to remove the application from the kubernetes, since the DNId is still set although it does not match this edge-node’s UUID.

Cluster Application Migrate (Automatic)

Controller Application Automatic Migrate

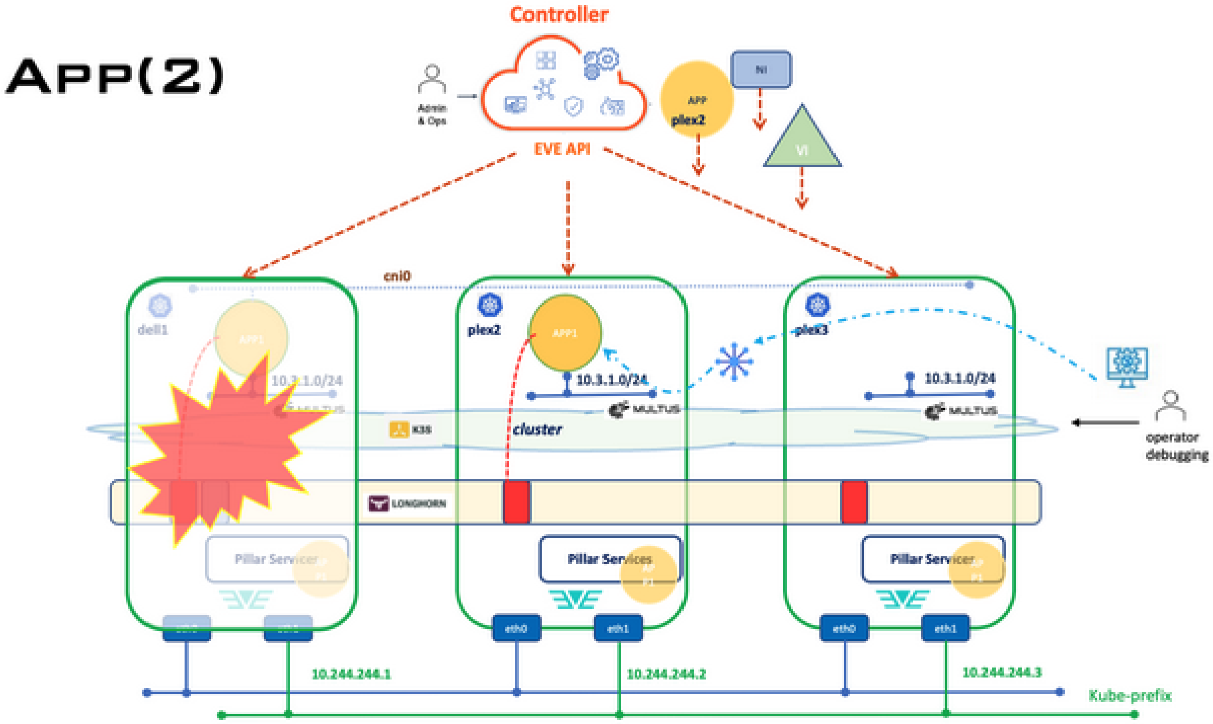
1. Controller monitors the info messages from all the edge-nodes in the cluster
2. When the Application is ‘AutoMigration’ set, if the Controller get an edge-node in the cluster reporting it is handling the running of the application on the edge-node, it needs to change the ‘DNId’ from the previous edge-node onto this new edge-node. Controller may need to verify either the previous edge-node is unreachable or get the info message from the previous edge-node reporting the application is no longer running on it (e.g. the edge-node is running in low memory condition).
3. ZInfoApp message added ‘ClusterAppRunning’ API, or a new info message can be defined for this purpose.

On-Site Application Automatic Migrate

- For the edge-node configured with the App Instance but with the DNId not matching itself, and if the App has the ‘AutoMigration’ set, it will monitor the kubernetes Pods in the user application namespace.

- When the kubernetes successfully migrate the Pod from a previous edge-node to this new node, it will set the ‘pod.Spec.NodeName’ to the node name of this edge-node.
- The new edge-node will report this through an info message, ‘ClusterAppRunning’ of ‘ZInfoApp’ message, to Controller and wait for Controller to change the DNId setting on the App Instance Configuration. (Note, the application is already running on this edge-node). It can periodically send this info message if without the DNId change to this edge-node (info message is not reliable)
- The reason the application runs before the DNId change is that the Application image is PVC based and exists through the longhorn CSI replication on all the edge-nodes. Same for the volume instance which is shared through the PVC replication. For ‘native container’, the image is downloaded to all the edge-nodes and uses the same named ‘containerd’ path on all the edge-nodes. The Network Instance name on all the edge-nodes is the same, although the Application may get a new IP address which depends on the IP configure mode (Detail in Cluster NI section).
- When the DNId changes are downloaded from Controller, the new edge-node changes the AppInstanceConfig.Activate to ‘true’ and runs through the normal App processing similar to the above ‘Manual Move’ part. The App metrics should start to work and the application status should display ‘Online’.
- On the previous edge-node of the application(s), the node may be down, or the node may be disconnected from the network. If it is the latter, we need to investigate the possibility of using the ‘CTRCTLl’ tools to remove the pod while without the access to the cluster after the specified ‘rescheduling’ timed out for the application(Since we know it is likely the application is being migrated onto another edge-node). If the application is not set for ‘AutoMigration’, we can have a fixed timeout value, say 30 minutes after the node lost connection to both the cluster and to the Controller.
- All the migration events will be logged, some key events can be event messages generated in the Controller to the cluster.

Cluster Application Migration Diagram



Adding new node to existing cluster

The following process is used to add a new node to an existing cluster. This is true for any node type:

- Additional control plane node (beyond the initial, or seed, node)
- First or additional worker node

Controller add or delete node to existing cluster

Need to plan to have nodes to be added or deleted, in particular one of the nodes in the cluster needs to be replaced.

- Let a user select a new edge-node (no different from the initial stage when 3 edge-nodes are selected).
- If it is an ‘Agent’ node, then need to set the IsAgent bool.

- When deleting a node, if the node is a ‘server’ node, and if the cluster is not fully up (with 3 nodes ready), may need to give warning to the user.
- All the NI, VI and Apps, should follow the same process and APIs as the original 3 server nodes
- If the new node is an ‘Agent’ node, there is no need to require the same port name on the Cluster-Interface as the case in all the server nodes.
- On the eve side, removal of a node from the cluster should follow kubernetes cordon/drain process to ensure longhorn can block the drain if the node-to-be-deleted contains the last replica.

On-Site add or delete node to existing cluster

- There should be no difference for any additional nodes or the initial 3 server nodes in terms of the operations on the edge-nodes. Assume the ‘same network’ of layer2 connectivity applies.

Cluster Metrics and Status

Every edge-node in the cluster will report its own App Metrics, for the VMI or Pods run on the edge-node. Controller can aggregate some of the metrics to be part of the cluster metrics display.
If there is a need to create new kubernetes cluster specific metrics for the cluster display, we can add those, and have the ‘Seed-Server’ to report those metrics. Each edge-node can also report their own view of cluster status (this is useful if the Seed-Server is down or the local network is partitioned)
It is needed for UI and ZCli to query about a cluster status, it probably should include this list:

- Cluster Name and UUID
- Seed-Server Name
- VIP IP address and which device owns it
- Total kubernetes nodes in ready state
- Each kubernetes node status (Ready or something else)
- Total Pods in ready state for each of the key components: Kube-System, Longhorn, KubeVirt, CDI and ‘eve-kube-app’
- Total Pods in non-ready state of the same components
- Memory and Storage stats of kubernetes cluster

Volume Instance Metrics and Status

Longhorn storage tracks the state of a volume into the following values:

- 1 creating
- 2 attached
- 3 detached
- 4 attaching
- 5 detaching
- 6 deleting

and the “robustness” of a volume into one of these values:

- 0 unknown
- 1 healthy
- 2 degraded (one or more replicas are offline)
- 3 faulted (volume is not available)

These two values should be read from longhorn metrics and possibly used to calculate the runstate of a volume instance. The last transition time of the volume robustness should be shown to convey the urgency of a degraded volume as a volume replicating towards being healthy for 1 hour is not as large of a worry as one which has been degraded for a week for example.

API Changes for Clustering

Edge-Node API

This is an example of the new ‘cluster’ API, is part of the device configuration of EVE API

```
message EdgeDevConfig {
...
    // cluster configuration
    EdgeNodeCluster cluster;
}

message EdgeNodeCluster {
    // cluster name, in case it has multiple cluster on the same site
    string cluster_name;
    // cluster UUID
    string cluster_id;
    // Cluster-Interface, for example "eth1"
    string cluster_interface;
    // The ‘cluster-prefix’ IP address of the ‘Cluster-Interface’, e.g. 10.244.244.2/28
    string cluster_ip_prefix;
    // This device is an ‘Agent’ node
    bool Is_agent;
    // Server IP address to join the cluster. E.g. 10.244.244.1
    string join_server_ip;
    // encrypted token string, use edge-node TMP to decrypt
    org.lfedge.eve.common.CipherBlock encrypted_cluster_token;
}
```

App Instance API

This is an example of the change of the ‘AppInstanceConfig’ API

```
message AppInstanceConfig {
…
    // This edge-node UUID for the Designated Node for the Application
   string designated_node_id;
…
}
```

Volume API

This is an example of the change of the ‘Volume’ API

```
message Volume {
…
    // To inform the edge-node if the device receives this Volume is
    // responsible to create volume, convert PVC or not
    string designated_node_id;
}
```

App Info Message API

This is an example of the change of the ‘Info’ API

```
enum ZInfoClusterNodeStatus {
   Z_INFO_CLUSTER_NODE_STATUS_UNSPECIFIED;
   Z_INFO_CLUSTER_NODE_STATUS_READY;          // cluster reports our node is ready
   Z_INFO_CLUSTER_NODE_STATUS_NOTREADY;  // cluster reports our node is ready
   Z_INFO_CLUSTER_NODE_STATUS_DOWN;         // cluster API server can not be reached
}

message ZInfoClusterNode {
    ZInfoClusterNodeStatus node_status;
}

message ZInfoMsg {
    oneof InfoContent {
       ...
       ZInfoClusterNode cluster_node;
    }
}

message ZInfoApp {
…
    // The App in cluster mode is currently running on this edge-node
    bool cluster_app_running;
}
```