Geo-redundancy

Product guide for prerelease

Document version 1841, produced on Friday, September 06, 2024.

main 90adc8bf40040649230176bbdd465f6261a2d8e0

Contents

List of Tables

1 Document History

Publication Date	Product Release	Notes
May 29, 2024	3.5.1	Updated ICE Server version reference to 3.5.41629.
April 15, 2024	3.5.0	Updated ICE Server version reference to 3.5.41160.
September 20, 2023	3.4.0	No updates.
July 27, 2023	3.3.0	Updated ICE Server version reference to 3.3.28975.
July 24, 2023	3.3.0	Updated version reference to 3.3.28856.
April 25, 2023	3.2.0	Updated ICE Server version reference to 3.2.26273.
February 17, 2023	3.2.0	Updated ICE Server version reference to 3.2.24516.
December 1, 2022	3.2.0	Release version number updates. Will be removing reference to kubespray, which is no longer supported. Also, much of the process and functionality described in this document is now handled by the 'ICE OS Configuration Wizard'. See the ***ICE Server Installation Guide*** for more information.
September 26, 2022	3.1.2	No updates.
April 18, 2022	3.1.1	Updated command for setting default storage class. Updated command to re-sychronize local Elasticsearch data.
March 15, 2022	3.1.0	Document created.

2 Tools

- This guide assumes that `helm` and `kubectl` are installed.
- The Instant Connect helm repo can be configured with

```
helm \
  repo \
  add \
  icet-helm \
  https://ic.repo.dillonkane.com:443/artifactory/icet-helm \
  --username instantconnect-artifactory \
```

```
  --password
      AKCp5bBgxtGQLmQrCBufrWBvDQLKQaZ2BGzpgCJLn8VNWBizdTtA539JgM9THuTUCr9ehgPxY
```

and

```
helm repo update
```

- We will use a tool called `kubemcsa` available here: https://github.com/admiraltyio/multicluster-service-account/releases/tag/v0.6.1

3 Prerequisites

Requirements to run Instant Connect Enterprise in a geo-redundant setting can be broken into two categories

3.1 Connectivity Requirements

- Pod IPs must be routable between all clusters
- Service IPs must be routable between all clusters
- DNS queries must work across clusters

3.2 Kubernetes Setup Requirements

These requirements are a result of the particular configuration that Instant Connect recommends for meeting the connectivity requirements above. If those connectivity requirements are met by another method, these may be unnecessary.

- Kubernetes must be configured to assign pod IP addresses from CIDRs allocated to each node
- Kubernetes must be configured to use the Flannel CNI plugin in “host-gw” mode
- All Kubernetes clusters must have unique, non-overlapping pod and service CIDRs

> To avoid conflicts with IPs already in use, we recommend updating the default IPs in CIDR (e.g., Kilo, WireGuard, docker0) before proceeding.

3.3 Planning

A well-planned configuration is key to ensuring a successful geo-redundant ICE installation. The configuration described below is provided as an example, and may or may not be appropriate for a given

installation. These values will be used throughout the document and will have to be substituted during an installation.

3.3.1 Cluster level

Table 2: Example Cluster IP Ranges

	DC1	DC2	Description
WireGuard Node IPs	10.80.0.0/16	10.110.0.0/16	Used to allocate IP addresses to the WireGuard network interface created on each node
Pod IPs	10.90.0.0/16	10.120.0.0/16	CIDR from which to allocate pod IPs
Service IPs	10.100.0.0/16	10.130.0.0/16	CIDR from which to allocate service IPs
DNS IP	10.100.0.10	10.130.0.10	IP of the cluster internal DNS service. Must be within the service CIDR
Kubernetes service IP	10.100.0.1	10.130.0.1	IP of the Kubernetes API within the cluster. Almost invariably the .1 of the service CIDR

3.3.2 Node Level

Values here are a consequence of choices made at the cluster level and are recoverable through various methods of introspection. Populating this table for each node will make subsequent configuration easier and is recommended.

Table 3: Example node information - 1

	dc1-1	Description
IP	192.168.1.105	IP address that would be reachable by other nodes, before any VPN setup
Cluster	DC1	Which cluster does this node belong to
Hostname	dc1-1	Hostname

	dc1-1	Description
Pod CIDR	10.90.0.0/24	The cluster's pod CIDR will be divided among the nodes of the cluster. Retrievable from the node spec like so `kubectl get node dc1_1 -o jsonpath='{.spec.podCIDR}'`
WireGuard IP	""	Placeholder, unknown until we setup Kilo and WireGuard
WireGuard public key	""	Placeholder, unknown until we setup Kilo and WireGuard

Table 4: Example node information - 2

	dc2-1	Description
IP	192.168.1.106	IP address that would be reachable by other nodes, before any VPN setup
Cluster	DC2	Which cluster does this node belong to
Hostname	dc2-1	Hostname
Pod CIDR	10.120.0.0/24	The cluster's pod CIDR will be divided among the nodes of the cluster. Retrievable from the node spec like so `kubectl get node dc2_1 -o jsonpath='{.spec.podCIDR}'`
WireGuard IP	""	Placeholder, unknown until we setup Kilo and WireGuard
WireGuard public key	""	Placeholder, unknown until we setup Kilo and WireGuard

3.3.3 Kubernetes Cluster Setup

3.3.3.1 Overview To setup our clusters, we'll be using Kubespray, checked out at git tag v2.15.0. See https://kubespray.io/ for basic instructions before proceeding.

> Proper time synchronization between all cluster nodes is critical, so the system administrator should verify that Network Time Protocol (NTP) is enabled and properly configured.

3.3.3.2 Installation

3.3.3.2.1 Setup Inventory

- Create your inventory files as normal. For example, `inventory/dc1/inventory.ini` looks like this:

```
[all]
dc1 ansible_host=192.168.1.105 ip=192.168.1.105

[kube-master]
dc1

[etcd]
dc1

[kube-node]
dc1

[calico-rr]

[k8s-cluster:children]
kube-master
kube-node
calico-rr
```

3.3.3.2.2 Required Options

File	Option	Value
`group_vars/k8s-cluster/k8s-cluster.yml`	`kube_network_plugin`	`flannel`

File	Option	Value
group_vars/k8s-cluster/k8s-cluster.yml	kube_service_addresses	10.100.0.0/16
group_vars/k8s-cluster/k8s-cluster.yml	kube_pods_subnet	10.90.0.0/16
group_vars/k8s-cluster/k8s-cluster.yml	skydns_server	10.100.0.10
group_vars/k8s-cluster/k8s-cluster.yml	kubeconfig_localhost*	**true**
group_vars/k8s-cluster/k8s-net-flannel.yml	flannel_backend_type	host-gw

*Optional, but useful

There are no other requirements or guidance on Kubespray options – appropriate configuration is situation specific.

3.3.3.2.3 Create Clusters

Create the clusters by

```
ansible-playbook \
  -i inventory/dc1/inventory.ini  \
  --become cluster.yml \
  -u exampleuser
```

Replace with values that are appropriate, add authentication and privilege escalation arguments as necessary for the situation. Setting up the user account (`exampleuser` in this example) with passwordless sudo permission is convenient but not required.

The resulting kubeconfig will be saved as `inventory/dc1/artifacts/admin.conf`. The geo instructions assume that this file will be located at `~/.kube/dc1`, so copy or adjust accordingly.

3.3.3.2.4 Setting Up Storage

Storage is a cluster-specific decision, but, for simplicity, we can use Rancher's local-path-provisioner.

Install the storage class and provisioner:

```
kubectl \
  apply \
  -f https://raw.githubusercontent.com/rancher/local-path-provisioner/
     master/deploy/local-path-storage.yaml
```

> NOTE: Indentation in a yaml file is significant. Do not alter the identation of the key/values in `local-path-storage.yaml`

For simplicity, set this storage class as the default:

```
kubectl \
  patch storageclass \
  local-path \
  -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-
     class":"true"}}}'
```

4 Installation

4.1 Overview

There will be several json files involved in this process, eventually leading to a folder populated like so:

```
| .
| ├── coredns
|     ├── coredns_dc1.json
|     └── coredns_dc2.json
| ├── ice
|     ├── ice_dc1.json
|     └── ice_dc2.json
| ├── kilo
|     ├── kilo_dc1_part1.json
|     ├── kilo_dc1_part2.json
|     ├── kilo_dc2_part1.json
|     └── kilo_dc2_part2.json
| └── seed
|     └── seed_dc2.json
```

4.2 Kilo/WireGuard

4.2.1 Overview

We use Kilo to create an overlay network that spans our clusters, with similar properties to the overlay networks we construct for ordinary Kubernetes clusters. Under the hood, this works by creating a WireGuard mesh between all nodes in our clusters.

4.2.2 Install

- If not done already, install WireGuard on every cluster node using your operating system's package manager. Specific instructions can be found here: https://www.wireguard.com/install/. It is not necessary to do any WireGuard setup such as key generation or interface creation at this time.
- Kilo tries to automatically find the correct endpoint to use for establishing connections, but can fail when the Docker shim is used in a cluster (the discovered endpoint ends up as the gateway of the Docker network). To ensure correctness, we can annotate all of our nodes with the correct IP (adjust for successive clusters):

```
export KUBECONFIG=~/.kube/dc1
for node in $(kubectl get nodes --no-headers | cut -d ' ' -f 1); do
  IP=$(kubectl \
    get node $node \
    -o jsonpath \
    --template "{.status.addresses[0].address}")
  kubectl annotate node $node kilo.squat.ai/force-endpoint=$IP:51820
  kubectl annotate node $node kilo.squat.ai/persistent-keepalive=10
done

export KUBECONFIG=~/.kube/dc2
for node in $(kubectl get nodes --no-headers | cut -d ' ' -f 1); do
  IP=$(kubectl \
    get node $node \
    -o jsonpath \
    --template "{.status.addresses[0].address}")
  kubectl annotate node $node kilo.squat.ai/force-endpoint=$IP:51820
  kubectl annotate node $node kilo.squat.ai/persistent-keepalive=10
done
```

- Populate `kilo/kilo_dc1_part1.json` like the following:

```
{
   "config": {
      "multidatacenter": {
```

```
            "wgCIDR": "10.80.0.0/16"
        },
        "peers": []
    }
}
```

Now run:

```
helm --kubeconfig ~/.kube/dc1 \
  upgrade -i \
  kilo-dc1 \
  instantconnect/ice-kilo \
  --version 2.0.10 \
  -f kilo/kilo_dc1_part1.json
```

- Repeat the process for DC2 by creating `kilo/kilo_dc2_part1.json`:

```
{
   "config": {
      "multidatacenter": {
         "wgCIDR": "10.110.0.0/16"
      },
      "peers": []
   }
}
```

Now run:

```
helm --kubeconfig ~/.kube/dc2 \
  upgrade -i \
  kilo-dc2 \
  instantconnect/ice-kilo \
  --version 2.0.10 \
  -f kilo/kilo_dc2_part1.json
```

4.2.3 Check install

Once the kilo pod in the kube-system namespace has successfully initialized, it will add annotations to all nodes in the cluster, viewable with

```
kubectl --kubeconfig ~/.kube/dc1 describe node | grep -A 20 "^Annotations"
```

Example:

```
Annotations:        kilo.squat.ai/endpoint: 192.168.1.105:51820
                    kilo.squat.ai/internal-ip: 192.168.1.105/24
                    kilo.squat.ai/key:
                        ___PUBLIC_KEY_OF_KILO0_INTERFACE_ON_DC1___
```

```
                    kilo.squat.ai/last-seen: 1612484404
                    kilo.squat.ai/wireguard-ip: 10.80.0.1/16
```

If any of those annotations are missing or the kilo pod is crashing, verify that WireGuard is correctly installed on all nodes and that the `wgCIDR` value is correctly populated.

You should also be able to ssh to a given node and run `sudo wg` and see output like this, matching the annotations:

```
$ sudo wg
interface: kilo0
  public key: ___PUBLIC_KEY_OF_KILO0_INTERFACE_ON_DC1___
  private key: (hidden)
  listening port: 51820
```

If your cluster has multiple nodes, they'll be represented in the output as "peers". Later, we'll be adding the nodes of any remote clusters as peers, but the objective here is simply to see that the interface is set up.

If the annotations are correct, repeat the steps for each cluster with the appropriate values.

Populate the WireGuard key and WireGuard IP values for each node in your tracking document.

4.2.4 Add Peers

Now we will create a mesh of the nodes between the cluster by creating "peer" objects for each node of each remote cluster. Expand on the content in `kilo/kilo_dc1_part1.json`, and save as `kilo/kilo_dc1_part2.json`

```
{
  "config": {
    "multidatacenter": {
      "wgCIDR": "10.80.0.0/16"
    },
    "peers": [
      {
        "name": "dc2-1",
        "allowedIPs": [
          "10.120.0.0/16",
          "10.110.0.1/32",
          "10.130.0.0/16"
        ],
        "endpoint": {
          "ip": "192.168.1.106",
          "port": 51820
        },
        "publicKey": "___PUBLIC_KEY_OF_KILO0_INTERFACE_ON_DC2___",
        "persistentKeepalive": 10
      }
    ]
  }
}
```

Perform the same with `kilo/kilo_dc2_part1.json`, and save as `kilo/kilo_dc2_part2.json`

```
{
  "config": {
    "multidatacenter": {
      "wgCIDR": "10.130.0.0/16"
    },
    "peers": [
      {
        "name": "dc1-1",
        "allowedIPs": [
          "10.90.0.0/16",
          "10.80.0.1/32",
          "10.100.0.0/16"
        ],
        "endpoint": {
          "ip": "192.168.1.105",
          "port": 51820
        },
        "publicKey": "___PUBLIC_KEY_OF_KILO0_INTERFACE_ON_DC1___",
        "persistentKeepalive": 10
      }
    ]
  }
}
```

Breaking down the peer object piece by piece:

Field	Value
name	It's easiest to use hostname here
allowedIPs	An array of: the portion of the pod CIDR assigned specifically to this node; the WireGuard IP of the node; and the full service CIDR.
endpoint	The non-WireGuard IP of the node and the default WireGuard port if 51820
publicKey	The public key of the node
persistentKeepalive	Optional, helps if one side of the connection is behind a NAT

There will be a peer object for each node in each remote cluster. In this example, each cluster only has 1 member.

Pay particularly close attention to the `AllowedIPs` section – most installation issues will come from misconfiguration of this field. The overall effect of `AllowedIPs` is twofold: first, when receiving

a packet, WireGuard will check if the source IP is allowed for the peer that sent it, if not the packet is dropped; second, when transmitting a packet to some destination IP, WireGuard will use the `AllowedIPs` field to identify which peer the packet should go to, encrypt it with the appropriate key, and send it to the appropriate endpoint.

In this example we have 1 peer, `dc2-1` with WireGuard listening at `192.168.1.106:51820`. We have 3 `AllowedIPs` CIDRs:

- `10.120.0.0/16`: The pod IPs that could be allocated to the specific remote node. Since this remote is a single-node cluster, we can just give it the entire range determined in our original setup and planning. In more complex scenarios, the allocated pod CIDR for a node is retrievable from the Kubernetes node object by means such as `kubectl describe node`.
- `10.110.0.1/32`: This is the IP of the WireGuard interface on the remote node. Recall that for dc2, we set the `wgCIDR` to `10.110.0.0/16`. Kilo then allocated an IP for each node's WireGuard interface from this range. This means that there is a 1:1 correspondence between this IP and the endpoint IP, hence the `/32`.
- `10.130.0.0/16`: This is the service IP range for the remote cluster. Since these IPs are not associated with any particular node, and there is no preferred node that they should go to, we include this for all nodes.

Once the configuration file is ready, create the peers on each cluster by upgrading with the `--set config.enablePeers=true` argument, replacing the references to the kubeconfig and `kilo_dc1_part2.json` as appropriate:

```
helm --kubeconfig ~/.kube/dc1 \
  upgrade -i \
  kilo-dc1 \
  instantconnect/ice-kilo \
  -f kilo/kilo_dc1_part2.json \
  --version 2.0.10 \
  --set config.enablePeers=true
```

Then perform the same with DC2:

```
helm --kubeconfig ~/.kube/dc2 \
  upgrade -i \
  kilo-dc2 \
  instantconnect/ice-kilo \
  -f kilo/kilo_dc2_part2.json \
  --version 2.0.10 \
  --set config.enablePeers=true
```

4.2.5 Check Peers

SSH to a node and run `sudo wg`. You should see some output like:

```
$ sudo wg
interface: kilo0
  public key: Y7tq+Npd9QzsmyVpDDvijkJ37fDd/VvUgpD74OXI0yo=
  private key: (hidden)
  listening port: 51820

peer: IS5x1SP1s7ilMultT/y/XPT4aFrHu7zXM9crqF5yKQQ=
  endpoint: 192.168.1.106:51820
  allowed ips: 10.110.0.1/32, 10.120.0.0/24, 10.130.0.0/16
  latest handshake: 3 seconds ago
  transfer: 348 B received, 404 B sent
```

If the “latest handshake” and “transfer” lines are absent, the peer is not connected. Try to ping an IP in the “allowed ips” range (such as 10.110.0.1 in this example) to force a connection and try again. Verify the connection on all nodes.

Similarly, there should also be connectivity to pods and services within the remote Kubernetes cluster. For example, from any of the nodes we have connectivity to the Kubernetes API of any of the clusters, by their cluster internal IP address

```
$ curl https://10.100.0.1 -k
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403

$ curl https://10.130.0.1 -k
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
```

4.3 CoreDNS

4.3.1 Overview

We use kubernetai, which is an external plugin for CoreDNS that allows DNS queries to fall through from the local cluster to successive clusters. This plugin is essentially the same as the standard Kubernetes CoreDNS plugin, the only real difference being that it allows specifying multiple clusters.

4.3.2 Install

First remove any existing CoreDNS deployments and services, if they exist

```
kubectl --kubeconfig ~/.kube/dc1 -n kube-system delete deploy coredns
kubectl --kubeconfig ~/.kube/dc1 -n kube-system delete svc kube-dns
```

Repeat for all clusters

> The particular name of the deployment and service will vary based on installation method, but will generally be some variant of `coredns` or `kube-dns`. *However*, `nodelocaldns` instances can remain – these are simply a caching mechanism.

> **WARNING**: This is critical, having both ordinary and multi-cluster DNS running simultaneously can lead to non-deterministic bootstrapping of subsequent components such as Cassandra – recovering from this can require specialized expertise.

Next, we need to provide the credentials necessary for each CoreDNS instance to query the remote Kubernetes APIs. For convenience, it's easiest to just replicate a complete set of credentials to all clusters like so:

```
export KUBECONFIG_A=~/.kube/dc1
export KUBECONFIG_B=~/.kube/dc2

# For DC1

export KUBECONFIG=$KUBECONFIG_A
kubemcsa export \
  --kubeconfig=$KUBECONFIG_A \
  -n kube-system \
  coredns \
  --as dc1 | kubectl -n kube-system apply -f -

kubemcsa export \
  --kubeconfig=$KUBECONFIG_B \
  -n kube-system \
  coredns \
  --as dc2 | kubectl -n kube-system apply -f -

# For DC2

export KUBECONFIG=$KUBECONFIG_B
kubemcsa export \
  --kubeconfig=$KUBECONFIG_A \
  -n kube-system \
  coredns \
  --as dc1 | kubectl -n kube-system apply -f -

kubemcsa export \
  --kubeconfig=$KUBECONFIG_B \
  -n kube-system \
  coredns \
  --as dc2 | kubectl -n kube-system apply -f -
```

This creates a `dc1` and `dc2` secret in the `kube-system` namespace of each cluster. If your clusters are named differently, adjust accordingly.

- Now create `coredns/coredns_dc1.json`:

```
{
  "config": {
    "multidatacenter": {
      "dcs": [
        {
          "name": "dc1",
          "kubernetesEndpoint": "https://10.100.0.1"
        },
        {
          "name": "dc2",
          "kubernetesEndpoint": "https://10.130.0.1"
        }
      ],
      "dnsIP": "10.100.0.10"
    }
  }
}
```

- and then create `coredns/coredns_dc2.json`:

```
{
  "config": {
    "multidatacenter": {
      "dcs": [
        {
          "name": "dc1",
          "kubernetesEndpoint": "https://10.100.0.1"
        },
        {
          "name": "dc2",
          "kubernetesEndpoint": "https://10.130.0.1"
        }
      ],
      "dnsIP": "10.130.0.10"
    }
  }
}
```

- The name *must* match what was fed into the kubemcsa commands above
- The `kubernetesEndpoint` is the internal Kubernetes service IP associated with that cluster. Note that while the Kubernetes API is customarily available on port 6443 externally, it listens on 443 inside the cluster network.
- All clusters can use the same `dcs` array, only `dnsIP` must be adjust for each cluster.

- Apply to each cluster with:

```
helm \
  --kubeconfig ~/.kube/dc1 \
  upgrade -i \
  coredns-dc1 \
  icet-helm/ice-coredns \
  --version 3.5.41629 \
  -f coredns/coredns_dc1.json

helm \
  --kubeconfig ~/.kube/dc2 \
  upgrade -i \
  coredns-dc2 \
  icet-helm/ice-coredns \
  --version 3.5.41629 \
  -f coredns/coredns_dc2.json
```

varying the kubeconfig and `coredns/coredns_dc1.json` references accordingly.

4.3.3 Check CoreDNS

We can setup a quick nginx pod in 1 cluster and attempt to connect to it from the other:

```
kubectl \
  --kubeconfig ~/.kube/dc1 \
  run \
  --image nginx \
  test-dc1 \
  --port 80 \
  --expose

kubectl \
  --kubeconfig ~/.kube/dc2 \
  run \
  -it --rm \
  --image busybox \
  busybox
```

This should leave you with a prompt inside the busybox container. Hit the nginx container like this:

```
wget -O- test-dc1
```

If successful, delete these pods and services

```
kubectl --kubeconfig ~/.kube/dc1 delete pod test-dc1`
kubectl --kubeconfig ~/.kube/dc1 delete svc test-dc1`
```

and repeat the process in the other direction. It's necessary to delete the first side so the (now) local nginx service doesn't mask the remote one.

4.4 Installing ICE

4.4.1 Overview

The process for installing ICE in a geo-redundant setting is slightly different than an ordinary installation. The configuration changes aside, the procedure now has to look something like this:

- Install ICE on some initial cluster – this cluster is only special for bootstrapping the initial installation and has no bearing on failure tolerance. In this document, that cluster is `DC1`
- This first ICE installation will hang, since it will be configured to expect some necessary credentials from all other clusters.
- Seed ICE components to all other clusters. This is done to create expected service accounts and objects.
- Retrieve the credentials from those clusters and load them into the first cluster using kubemcsa again
- Allow the ICE installation to finish on the first cluster
- Ensure that Cassandra has accepted and propagated the schema updates to all clusters
- Iterate through and complete all seeded installations
- Test Cassandra replication
- Test Kafka replication
- Test ICE client behavior

4.4.2 Install ICE in initial cluster

- Create a minimal `ice/ice_dc1.json`. Please refer to Appendix A for a full sample.

```
{
  "charts": {
    "iceMinio": {
      "enabled": true,
      "values": {
        "multidatacenter": {
          "enabled": true,
          "currentDC": "dc1",
          "remoteDC": "dc2"
        }
      }
    },
```

```
    "rallypoint": {
      "enabled": true,
      "values": {
        "rallypoint": {
          "name": "dc1",
          "peerMeshes": [
            "ice-rallypoint-discovery-dc2"
          ]
        },
        "patch": {
          "agent": {
            "patchAgentName": "Default Patch Agent - dc1"
          }
        },
        "reflector": {
          "name": "Default Reflector - dc1",
          "config": {
            "multicastInterfaceName": "eth0"
          }
        }
      }
    },
    "iceCassandra": {
      "enabled": true,
      "values": {
        "multi-casskop": {
          "k8s": {
            "local": "dc1",
            "remote": [
              "dc2"
            ]
          }
        }
      }
    },
    "instantConnectEnterprise": {
      "enabled": true,
      "spec": {
        "timeout": 10800
      },
      "values": {
        "config": {
          "include": {
            "multiCassKop": true
          },
          "multidatacenter": {
            "enabled": true,
            "currentDC": "dc1",
            "dcs": [
              {
                "name": "dc1"
```

```
                },
                {
                  "name": "dc2"
                }
              ]
            }
          }
        }
      }
    }
  }
```

- This is a minimal example for geo-redundancy, include additional options as desired.
- Going item by item:

rallypoint	Description
rallypoint.name	This is used to create a distinct service for use in discovery and meshing. The example above results in a service named `ice-rallypoint-discovery-dc1`.
rallypoint.peerMeshes	An array of rallypoints to form a mesh with. These are discovered by simple DNS lookups.
reflector.name	A string name to display for the default reflectors.
reflector.config.multicastInterfaceName	The name of the multicast interface the reflector should bind to.

iceCassandra	Description
local	Used for naming datacenters within the Cassandra cluster.
remote	An array of remote Kubernetes clusters. Corresponds to the name of the secret expected by the Cassandra operator *and* the name of the resulting Cassandra datacenter.

instantConnectEnterprise	Description
multiCassKop	Responsible for Cassandra initialization and reconciliation across all clusters. Should only be enabled in 1.
currentDC	Used in conjunction with `dcs` array for orchestrating the various replication and coordination mechanisms
dcs	An array (of maps, not strings) of all clusters expected to be part of the geo-redundant system.

- Apply this to the initial cluster with:

```
helm \
  --kubeconfig ~/.kube/dc1 \
  -n ice-release \
  upgrade -i \
  --create-namespace \
  ice-helm-operator \
  icet-helm/ice-helm-operator-develop \
  --version 3.5.41629
  -f ice/ice_dc1.json
```

4.4.3 Seed ICE Installation on all other clusters

Create `seed/seed_dc2.json`:

```
{
  "charts": {
    "iceMinio": {
      "enabled": true,
      "values": {
        "multidatacenter": {
          "enabled": true,
          "currentDC": "dc2",
          "remoteDC": "dc1"
        }
      }
    },
    "rallypoint": {
      "values": {
        "rallypoint": {
          "name": "dc2",
          "peerMeshes": [
            "ice-rallypoint-discovery-dc1"
          ]
```

```
        },
        "patch": {
          "agent": {
            "patchAgentName": "Default Patch Agent - dc2"
          }
        },
        "reflector": {
          "name": "Default Reflector - dc2"
        }
      }
    },
    "iceCassandra": {
      "values": {
        "multi-casskop": {
          "enabled": false
        }
      }
    },
    "instantConnectEnterprise": {
      "enabled": false
    }
  }
}
```

Apply to any remaining clusters:

```
helm \
  --kubeconfig ~/.kube/dc2 \
  -n ice-release \
  upgrade -i \
  --create-namespace \
  ice-helm-operator \
  icet-helm/ice-helm-operator-develop \
  --version 3.5.41629 \
  -f seed/seed_dc2.json
```

4.4.4 Resume Initial Installation

> **NOTE** Use `watch kubectl get ns` to make sure `ice-cassandra` namespace appears before performing steps in this section.

```
export KUBECONFIG_A=~/.kube/dc1
export KUBECONFIG_B=~/.kube/dc2

export KUBECONFIG=$KUBECONFIG_A
kubemcsa export \
    --kubeconfig=$KUBECONFIG_A \
    --namespace ice-cassandra cassandra-operator \
    --as dc1 \
    | kubectl -n ice-cassandra apply -f -

kubemcsa export \
    --kubeconfig=$KUBECONFIG_B \
    --namespace ice-cassandra cassandra-operator \
    --as dc2 \
    | kubectl -n ice-cassandra apply -f -

export KUBECONFIG=$KUBECONFIG_B
kubemcsa export \
    --kubeconfig=$KUBECONFIG_A \
    --namespace ice-cassandra cassandra-operator \
    --as dc1 \
    | kubectl -n ice-cassandra apply -f -

kubemcsa export \
    --kubeconfig=$KUBECONFIG_B \
    --namespace ice-cassandra cassandra-operator \
    --as dc2 \
    | kubectl -n ice-cassandra apply -f -
```

We should now see the Cassandra operator start and begin to start the Cassandra nodes for each cluster. This is a sequential operation and may take some time, on the order of 5-10 minutes.

Once Cassandra is fully started, the Instant Connect Enterprise installation in the first cluster will complete.

4.4.5 Check Initial Instant Connect Enterprise Installation

First, verify that modelmanager has finished by checking the `ice-arcus` namespace:

```
$ kubectl -n ice-arcus get pods

NAME                                    READY   STATUS      RESTARTS
    AGE
```

```
client-bridge-764b4664bb-wmv6t              1/1     Running     0
    50m
elasticsearch-arcus-es-member-0             1/1     Running     0
    50m
modelmanager-git-22fcd93-8fncc              0/1     Completed   0
    50m
platform-services-6949c8c57f-pngqw          1/1     Running     0
    50m
remote-installer-7bc7fdc5f7-nbrgb           1/1     Running     0
    50m
rest-bridge-7c955446dd-z89c2                1/1     Running     0
    50m
server-bridge-78dcf556c4-rcq7f              1/1     Running     0
    50m
swagger-ui-bf6965559-nwkfg                  1/1     Running     0
    50m
telephony-registration-54fdb7554d-q7wfr     1/1     Running     0
    50m
```

Correct output is for the modelmanager pod to be `Completed` and all other pods to be `Running`

- Similarly, all pods in the `ice-rallypoint` namespace should be `Running`. Delete any pods that are not running and allow them to attempt to reinitialize.
- Check that Cassandra reports that it's healthy

  > **Note:** If there is more than one Cassandra pod, you must repeat all `nodetool` commands for each, one after another.

  A correct response at DC1 looks something like this:

```
$ kubectl \
  --kubeconfig ~/.kube/dc1 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc1-rack1-0 \
   -- nodetool status

Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load       Tokens       Owns (effective)  Host ID
                                  Rack
UN  10.90.0.51   389 KiB    256          100.0%            9ea2339d-3cc4
    -4970-af4f-39c7ffbd9508  rack1
```

A correct response at DC2 looks something like this:

```
$ kubectl \
  --kubeconfig ~/.kube/dc2 \
```

```
  -n ice-cassandra -c cassandra exec -it \
  ice-dc2-rack1-0 \
  -- nodetool status

Datacenter: dc2
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load       Tokens       Owns (effective)  Host ID
                                Rack
UN  10.120.0.31  472.56 KiB  256          100.0%            39fe4a96-7b39
    -45c9-9a8b-88f63224814d  rack1
```

The UN (Up, Normal) is most critical. Other items such as IP address and datacenter name should match expectations.

- Check that Cassandra has a consistent schema:

Healthy output looks like this in DC1:

```
$ kubectl \
  --kubeconfig ~/.kube/dc1 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc1-rack1-0 \
  -- nodetool describecluster

Cluster Information:
    Name: ice
    Snitch: org.apache.cassandra.locator.GossipingPropertyFileSnitch
    DynamicEndPointSnitch: enabled
    Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
    Schema versions:
        4ca0c366-fd08-3745-8950-a585d6c80f9b: [10.120.0.31, 10.90.0.51]
```

Repeat the same check in DC2:

```
kubectl \
  --kubeconfig ~/.kube/dc2 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc2-rack1-0 \
  -- nodetool describecluster
```

All nodes should be listed in the array after a single schema version. If there is more than one schema version listed, restart the Cassandra pods one-by-one, waiting for each pod to become fully ready before restarting the next. The startup process will force a reconciliation of schema version.

- Finally, check that data has replicated correctly, by running a command like this against each cluster. Note that the cluster name is injected into the pod name and adjust accordingly.

```
$ kubectl \
  --kubeconfig ~/.kube/dc1 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc1-rack1-0 \
  -- cqlsh -e "select count(*) from dev.changeset;"

Warning: Cannot create directory at `/home/cassandra/.cassandra`. Command
    history will not be saved.


 count
-------
   107

(1 rows)
```

All clusters should provide the same output. For our purposes here, all that matters is that all clusters agree – the exact number is irrelevant.

```
kubectl \
  --kubeconfig ~/.kube/dc2 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc2-rack1-0 \
  -- cqlsh -e "select count(*) from dev.changeset;"
```

If a cluster says 0, that suggests that there was an issue with the bootstrapping process. Verify that connectivity and DNS is working as it should in all clusters.

- Since consistency is so critical for us at this stage, we might as well also run a `nodetool repair` on each Cassandra node before proceeding, starting with DC1:

```
kubectl \
  --kubeconfig ~/.kube/dc1 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc1-rack1-0 \
  -- nodetool repair
```

then DC2

```
kubectl \
  --kubeconfig ~/.kube/dc2 \
  -n ice-cassandra -c cassandra exec -it \
  ice-dc2-rack1-0 \
  -- nodetool repair
```

4.4.6 Override random initial password

Some Instant Connect Enterprise services internal to the cluster need a credential to operate. We generate this randomly, so it's easy enough to just copy from the first cluster to all other clusters.

Retrieve the password (values in Kubernetes secrets are base64 encoded, so we retrieve and apply the value as-is, rather than decoding and re-encoding)

```
$ kubectl \
  --kubeconfig ~/.kube/dc1 \
  -n ice-arcus \
  get secrets \
  init-superuser-pass \
  -o=jsonpath='{.data.pass}' ; echo

aBcDeFgHiJkLmN==
```

Then, patch all other clusters to match:

```
$ kubectl \
  --kubeconfig ~/.kube/dc2 \
  -n ice-arcus \
  patch \
  secret \
  init-superuser-pass \
  -p '{"data":{"pass":"aBcDeFgHiJkLmN=="}}'
```

4.4.7 Complete ICE installation on all other clusters

Now we'll finalize the installation of ICE on clusters. Create a minimal `ice/ice_dc2.json`. Please refer to Appendix C for a full sample.

```
{
  "charts": {
    "iceMinio": {
      "enabled": true,
      "values": {
        "multidatacenter": {
          "enabled": true,
          "currentDC": "dc2",
          "remoteDC": "dc1"
        }
      }
    },
    "rallypoint": {
      "values": {
        "rallypoint": {
          "name": "dc2",
```

```
          "peerMeshes": [
            "ice-rallypoint-discovery-dc1"
          ]
        },
        "patch": {
          "agent": {
            "patchAgentName": "Default Patch Agent - dc2"
          }
        },
        "reflector": {
          "name": "Default Reflector - dc2",
          "config": {
            "multicastInterfaceName": "eth0"
          }
        }
      }
    },
    "iceCassandra": {
      "values": {
        "multi-casskop": {
          "enabled": false
        }
      }
    },
    "instantConnectEnterprise": {
      "spec": {
        "timeout": 10800
      },
      "values": {
        "config": {
          "include": {
            "multiCassKop": false
          },
          "multidatacenter": {
            "enabled": true,
            "currentDC": "dc2",
            "dcs": [
              {
                "name": "dc1"
              },
              {
                "name": "dc2"
              }
            ]
          }
        }
      }
    }
  }
}
```

Similar to the `ice/ice_dc1.json`, specify any other desired options. Note that, while this is broadly similar to `ice/ice_dc1.json`, fields indicating current cluster or remote endpoints may be inverted, as here:

```
      rallypoint:
        name: dc2
        peerMeshes:
          - ice-rallypoint-discovery-dc1
      reflector:
        name: Default Reflector - dc2
```

and here:

```
        multidatacenter:
          currentDC: dc2
          dcs:
            - name: dc1
            - name: dc2
          enabled: true
```

Lastly, note that `multicasskop` remains disabled – that is only required in the first cluster.

- Apply the configuration:

```
helm \
  --kubeconfig ~/.kube/dc2 \
  -n ice-release \
  upgrade -i \
  --create-namespace \
  ice-helm-operator \
  icet-helm/ice-helm-operator-develop \
  --version 3.5.41629 \
  -f ice/ice_dc2.json
```

- Wait for modelmanager to complete as on the previous cluster

4.4.8 Check ICE installation on all clusters

- Check pods in the `ice-rallypoint` namespace in all clusters, restart any that aren't `Running`
- There can be issues with the Kafka replication components deploying faster than Kafka becomes configured to their expectations. It is a good idea to restart them at this point as a precaution. These issues come about from the existence or non-existence of the various Kafka topics at startup, so they are confined to installation. On each cluster, run:

```
kubectl \
  -n ice-kafka \
  delete pod \
  -l run=ksqldb

kubectl \
  -n ice-kafka \
  delete pod \
  -l app.kubernetes.io/name=kafka-mirror-maker-2
```

5 Data re-synchronization after a Data Center outage

In a Geo Redundancy setup, if the data center outage lasts:

- **Less than 1 hour:** The data re-synchronization takes place automatically once the clusters in both data centers are communicating again. The following steps are strongly recommended for validation purposes.
- **1-2 hours:** The data re-synchronization takes place automatically once the clusters in both data centers are communicating again. However, the data re-synchronization time can be shortened substantially by following the recovery steps below.
- **More than 2 hours:** The recovery steps below *must* be performed to ensure data is in sync between the two data centers.

> Note: Do *not* skip any of the recovery steps below. Each section of the process *must* be performed.

5.1 Re-sychronize Kafka

Replication failure between clusters will break dynamic behaviors of the ICE clients, such as: - Channels being added to a user's dashboard on membership change - Private call notifications not appearing or not being responsive - Abrupt system reboots

5.1.1 Check MirrorMaker2 status

1. The MirrorMaker2 pod takes ~10 minutes to reach a 'RUNNING' status. If it fails to start within this timeframe, then restart the pod.
2. Once the MirrorMaker2 pod on DC1 shows a 'RUNNING' status: `kubectl -n ice-kafka get kafkamirrormaker2s.kafka.strimzi.io mm-dc2 -o=jsonpath='{.status}'| jq .`

3. The 'Conditions' state should be 'True', while the 'Connectors' state should be 'RUNNING'.
4. Once the MirrorMaker2 pod on DC2 shows 'RUNNING' status:

   ```
   kubectl -n ice-kafka get kafkamirrormaker2s.kafka.strimzi.io mm-dc1 -o=jsonpath='{.status}'| jq .
   ```

5. The 'Conditions' state should be 'True', while the 'Connectors' state should be 'RUNNING'.
6. In this event, restart the MirrorMaker2 pod on DC1 first, once that DC1 pod is running, *then* restart the MirrorMaker2 pod on DC2.

5.2 Re-sychronize Cassandra database

> **Note:** If there is more than one Cassandra pod, you must perform `nodetool repair` on each pod, one after another.

Synchronization failure between Cassandra sites will manifest as inconsistency between sites (channel membership differences, license status, etc.). The most common cause of this is a network partition that Cassandra has not been able to automatically reconcile. In this scenario, run `nodetool repair` on each Cassandra node in every cluster, one after another. The first Cassandra node in cluster DC1 can be repaired using this command:

```
kubectl \
  --kubeconfig ~/.kube/dc1 \
  -n ice-cassandra exec -it \
  ice-dc1-rack1-0 \
  -- nodetool repair
```

Adjust the command as necessary for the remaining nodes.

Note that Cassandra repairs may cause interruptions of service. In ICE 2.0.0, this is a manual process. For ICE 2.1.0 and later, though, a utility is included that continuously repairs subsections of the database as needed, reducing or eliminating the need for manual repairs.

5.3 Re-sychronize local Elasticsearch data

Elasticsearch may fall out of sync in two ways. The first is for Elasticsearch to fall out of sync with Cassandra. Elasticsearch can be manually re-synchronized with Cassandra by running:

```
kubectl \
  -n ice-arcus \
  create job \
  resync-$(date "+%Y%m%d-%H%M") \
```

```
  --from=cronjob/elastic-sync-dc1

kubectl \
  -n ice-arcus \
  create job \
  resync-$(date "+%Y%m%d-%H%M") \
  --from=cronjob/elastic-sync-dc2
```

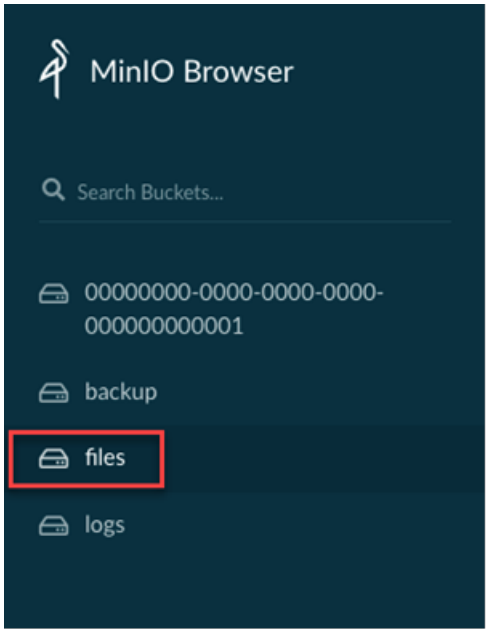
Adding a date reference is not strictly required, but helps with organization.

This forced resynchronization is also run automatically every day by default.

If Elasticsearch still fails to update, this indicates that this is actually a Kafka replication problem.

5.4 MinIO Sync Failure

By default, only the `files` directory is replicated between the data centers:



The directory content is automatically resynchronized after a data center outage. In the very rare case that a full resynchronization is required, follow the process below.

1. Redefine MinIO mirroring on DC1:
    - Determine the replicaset's name:

      ```
      kubectl --kubeconfig ~/.kube/dc1 -n ice-minio get replicasets
      ```

- Delete the replicaset that corresponds to the Minio mirroring, with `minio-mirror-` prefix:

```
kubectl --kubeconfig ~/.kube/dc1 -n ice-minio delete replicaset
    minio-mirror-....
```

2. Redefine MinIO mirroring on DC2:

   - Determine the replicaset's name:

```
kubectl --kubeconfig ~/.kube/dc2 -n ice-minio get replicasets
```

   - Delete the replicaset that corresponds to the Minio mirroring, with `minio-mirror-` prefix:

```
kubectl --kubeconfig ~/.kube/dc2 -n ice-minio delete replicaset
    minio-mirror-....
```

3. Confirm the replication is running properly by reviewing the Minio Mirror logs while the directory content is updated:

```
kubectl --kubeconfig ~/.kube/dc1 -n ice-minio logs -f -l app=minio-
    mirror
kubectl --kubeconfig ~/.kube/dc2 -n ice-minio logs -f -l app=minio-
    mirror
```

6 Upgrading ICE Server

When upgrading ICE Server, there are some additional steps for geo-redundancy:

> **Notes:**
>
> - Please refer to the latest ***ICE Server Upgrade Guide*** for information on upgrading ICE Server.
> - We recommend an ad hoc backup of the server prior to beginning troubleshooting or upgrade processes. To create an ad hoc backup, please see the ***Server Administration Guide***, *Appendix A , Ad Hoc Server Backup*.

1. Upgrade ICE Server on one cluster. It does not matter which cluster is upgraded first.
2. When done, follow the steps in the *'Check Initial Instant Connect Enterprise Installation'* section of this document.
3. Upgrade ICE Server on the remaining cluster. This should go noticeably faster, since some of the work was already done by the first cluster's upgrade.
4. When done, follow the steps in the *'Check ICE installation on all clusters'* section of this document.

7 Appendix A: Full Sample `ice_dc1.json`

The following is a full sample `ice/ice_dc1.json` that may be used for installation with cluster FQDN name (for enabling SSL/TLS access) and Geo redundancy. Underscored '___' values in the sample below must be replaced with the actual values provided during ICE installation. Please review the ICE Server Administration Guide for allowed values.

```
{
  "charts": {
    "iceMinio": {
      "enabled": true,
      "values": {
        "multidatacenter": {
          "enabled": true,
          "currentDC": "dc1",
          "remoteDC": "dc2"
        },
        "config": {
          "ingress": {
            "hosts": [
              "___DC1_FQDN___"
            ]
          }
        }
      }
    },
    "iceLogging": {
      "enabled": false
    },
    "iceIngress": {
      "enabled": true,
      "values": {
        "host": "___DC1_FQDN___"
      }
    },
    "iceMonitoring": {
      "enabled": true,
      "values": {
        "grafana": {
          "grafana.ini": {
            "server": {
              "domain": "___DC1_FQDN___"
            },
            "smtp": {
              "enabled": true,
              "from_address": "___FULL_EMAIL_ADDRESS___",
              "from_name": "___EMAIL_USER_COMMON_NAME___",
              "host": "___SMTP_SERVER___:___SMTP_SERVER_PORT___",
              "user": "apikey",
```

```
            "password": "___EMAIL_API_KEY___",
            "skip_verify": true
          }
        },
        "ingress": {
          "enabled": true,
          "hosts": [
            "___DC1_FQDN___"
          ]
        }
      },
      "prometheus": {
        "server": {
          "retention": "7d"
        }
      },
      "loki": {
        "config": {
          "table_manager": {
            "retention_deletes_enabled": true,
            "retention_period": "7d"
          }
        }
      }
    }
  },
  "rallypoint": {
    "values": {
      "rallypoint": {
        "name": "dc1",
        "peerMeshes": [
          "ice-rallypoint-discovery-dc2"
        ]
      },
      "patch": {
        "agent": {
          "patchAgentName": "Default Patch Agent - dc1"
        }
      },
      "reflector": {
        "name": "Default Reflector - dc1",
        "config": {
          "multicastInterfaceName": "___MULTICAST_INTERFACE_NAME___"
        }
      }
    }
  },
  "iceCassandra": {
    "values": {
      "multi-casskop": {
        "k8s": {
```

```
            "local": "dc1",
            "remote": [
              "dc2"
            ]
          }
        }
      }
    },
    "instantConnectEnterprise": {
      "spec": {
        "timeout": 10800
      },
      "values": {
        "config": {
          "ingress": {
            "enabled": true,
            "hosts": [
              "___DC1_FQDN___"
            ]
          },
          "swaggerUI": {
            "enabled": false,
            "json": {
              "merge.json": {
                "servers": [
                  {
                    "description": "public rest endpoint",
                    "url": "https://___DC1_FQDN___/instant-connect"
                  }
                ]
              }
            },
            "mergeJson": true
          },
          "include": {
            "multiCassKop": true
          },
          "cassandra": {
            "nCassandraNodes": 1,
            "replicationFactor": 1
          },
          "multidatacenter": {
            "enabled": true,
            "currentDC": "dc1",
            "dcs": [
              {
                "name": "dc1"
              },
              {
                "name": "dc2"
              }
```

```
            ]
          }
        },
        "devInclude": {
          "IPPhoneBridge": true
        },
        "env": {
          "arcusComponents": {
            "ipphone-bridge": {
              "redirect.base.url": "https://___DC1_FQDN___/ipphone",
              "static.resource.server.url":
                "https://___DC1_FQDN___/ipphone"
            }
          }
        }
      }
    },
    "sonobuoy": {
      "enabled": false
    }
  }
}
```

8 Appendix B: Full Sample `seed_dc2.json`

The following is a full sample `seed/seed_dc2.json`. Underscored '___' values in the sample below must be replaced with the actual values provided during ICE installation. Please review the ICE Server Administration Guide for allowed values.

```
{
  "charts": {
    "iceMinio": {
      "enabled": true,
      "values": {
        "multidatacenter": {
          "enabled": true,
          "currentDC": "dc2",
          "remoteDC": "dc1"
        }
      }
    },
    "rallypoint": {
      "values": {
        "rallypoint": {
          "name": "dc2",
          "peerMeshes": [
            "ice-rallypoint-discovery-dc1"
          ]
```

```
        },
        "patch": {
          "agent": {
            "patchAgentName": "Default Patch Agent - dc2"
          }
        },
        "reflector": {
          "name": "Default Reflector - dc2",
          "config": {
            "multicastInterfaceName": "___MULTICAST_INTERFACE_NAME___"
          }
        }
      }
    },
    "iceMonitoring": {
      "enabled": false
    },
    "iceCassandra": {
      "values": {
        "multi-casskop": {
          "enabled": false
        }
      }
    },
    "instantConnectEnterprise": {
      "enabled": false
    }
  }
}
```

9 Appendix C: Full Sample `ice_dc2.json`

The following is a full sample `ice/ice_dc2.json` that may be used for installation with cluster FQDN name (for enabling SSL/TLS access) and Geo redundancy. Underscored '___' values in the sample below must be replaced with the actual values provided during ICE installation. Please review the ICE Server Administration Guide for allowed values.

```
{
  "charts": {
    "iceMinio": {
      "enabled": true,
      "values": {
        "multidatacenter": {
          "enabled": true,
          "currentDC": "dc2",
          "remoteDC": "dc1"
        },
```

```
        "config": {
          "ingress": {
            "hosts": [
              "___DC2_FQDN___"
            ]
          }
        }
      }
    },
    "iceLogging": {
      "enabled": false
    },
    "iceIngress": {
      "enabled": true,
      "values": {
        "host": "___DC2_FQDN___"
      }
    },
    "iceMonitoring": {
      "enabled": true,
      "values": {
        "grafana": {
          "grafana.ini": {
            "server": {
              "domain": "___DC2_FQDN___"
            },
            "smtp": {
              "enabled": true,
              "from_address": "___FULL_EMAIL_ADDRESS___",
              "from_name": "___EMAIL_USER_COMMON_NAME___",
              "host": "___SMTP_SERVER___:___SMTP_SERVER_PORT___",
              "user": "apikey",
              "password": "___EMAIL_API_KEY___",
              "skip_verify": true
            }
          },
          "ingress": {
            "enabled": true,
            "hosts": [
              "___DC2_FQDN___"
            ]
          }
        },
        "prometheus": {
          "server": {
            "retention": "7d"
          }
        },
        "loki": {
          "config": {
            "table_manager": {
```

```
            "retention_deletes_enabled": true,
            "retention_period": "7d"
          }
        }
      }
    }
  },
  "rallypoint": {
    "values": {
      "rallypoint": {
        "name": "dc2",
        "peerMeshes": [
          "ice-rallypoint-discovery-dc1"
        ]
      },
      "patch": {
        "agent": {
          "patchAgentName": "Default Patch Agent - dc2"
        }
      },
      "reflector": {
        "name": "Default Reflector - dc2",
        "config": {
          "multicastInterfaceName": "___MULTICAST_INTERFACE_NAME___"
        }
      }
    }
  },
  "iceCassandra": {
    "values": {
      "multi-casskop": {
        "enabled": false
      }
    }
  },
  "instantConnectEnterprise": {
    "spec": {
      "timeout": 10800
    },
    "values": {
      "config": {
        "ingress": {
          "enabled": true,
          "hosts": [
            "___DC2_FQDN___"
          ]
        },
        "swaggerUI": {
          "enabled": false,
          "json": {
            "merge.json": {
```

```
                "servers": [
                  {
                    "description": "public rest endpoint",
                    "url": "https://___DC2_FQDN___/instant-connect"
                  }
                ]
              }
            },
            "mergeJson": true
          },
          "include": {
            "multiCassKop": false
          },
          "cassandra": {
            "nCassandraNodes": 1,
            "replicationFactor": 1
          },
          "multidatacenter": {
            "enabled": true,
            "currentDC": "dc2",
            "dcs": [
              {
                "name": "dc1"
              },
              {
                "name": "dc2"
              }
            ]
          }
        },
        "devInclude": {
          "IPPhoneBridge": true
        },
        "env": {
          "arcusComponents": {
            "ipphone-bridge": {
              "redirect.base.url": "https://___DC2_FQDN___/ipphone",
              "static.resource.server.url":
                "https://___DC2_FQDN___/ipphone"
            }
          }
        }
      }
    },
    "sonobuoy": {
      "enabled": false
    }
  }
}
```