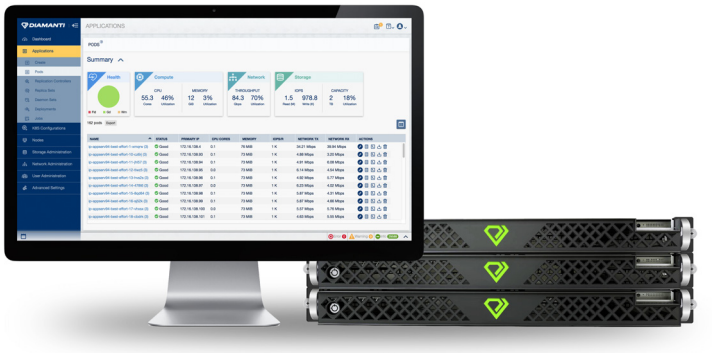
Diamanti D-Series

v2.3.0 Quick Start Guide



DIAMANTI

Diamanti D-Series v2.3.0 Quick Start Guide, First Edition, January 2020

Contents

Getting Started

Managing the D-Series Appliance

1

Getting Started

Welcome to the *Diamanti D-Series Quick Start Guide*. This guide introduces the Diamanti D-Series appliance and provides an overview of the principal components and features of the Diamanti software.

This guide then describes how to perform the most common operations, including how to create a cluster, define Diamanti objects such as networks and storage volumes, and deploy workloads using Kubernetes.

Note: For information about installing and configuring the Diamanti D-Series appliance, see the *Diamanti D-Series Installation Guide*. Refer to the *Diamanti D-Series Command Line Interface Guide* for details about the command line interface. For more information about the Diamanti User Interface (UI), see the *Diamanti D-Series User Interface Guide*.

This document is designed for the following users:

- Cluster administrators
- User administrators
- Storage administrators
- Network administrators
- Developers

Cluster, storage, and network administrators ensure the proper deployment and daily operation of an organization's IT infrastructure, including Diamanti D-Series appliances. Developers, meanwhile, deploy containers on Diamanti appliances.

Introducing the Diamanti D-Series Appliance

Diamanti provides a hyperconverged platform for running Linux containers that integrates with open source software including Docker and Kubernetes. The D-Series appliance, along with Diamanti software, offers developers and administrators on-demand access to fast, predictable data services. This allows developers to specify I/O policies on a per-container or per-application basis, as needed.

These policies link standard applications such as databases (for example, MongoDB, Cassandra, MySQL, and Kafka), networking services (such as NGINX), and CI/CD applications (for instance, Jenkins) with software-defined network and storage resources, ensuring that the I/O requirements for each application are fulfilled.

This provides administrators with an easy-to-use, standards-based container platform.

Understanding D-Series Appliance Basics

The Diamanti D-Series appliance is purpose-built for containerized applications that combines a hyperconverged infrastructure with the performance and efficiency of bare-metal containers. The Diamanti software provides a virtualized, clustered approach to network and storage.

Together, this provides containerized workloads with guaranteed network and storage performance to complement the CPU and memory allocation provided by Linux containers. These guarantees allow more workloads to run on the same class of hardware without sacrificing application quality, reliability, or performance.

The following illustrates the Diamanti D-Series architecture:

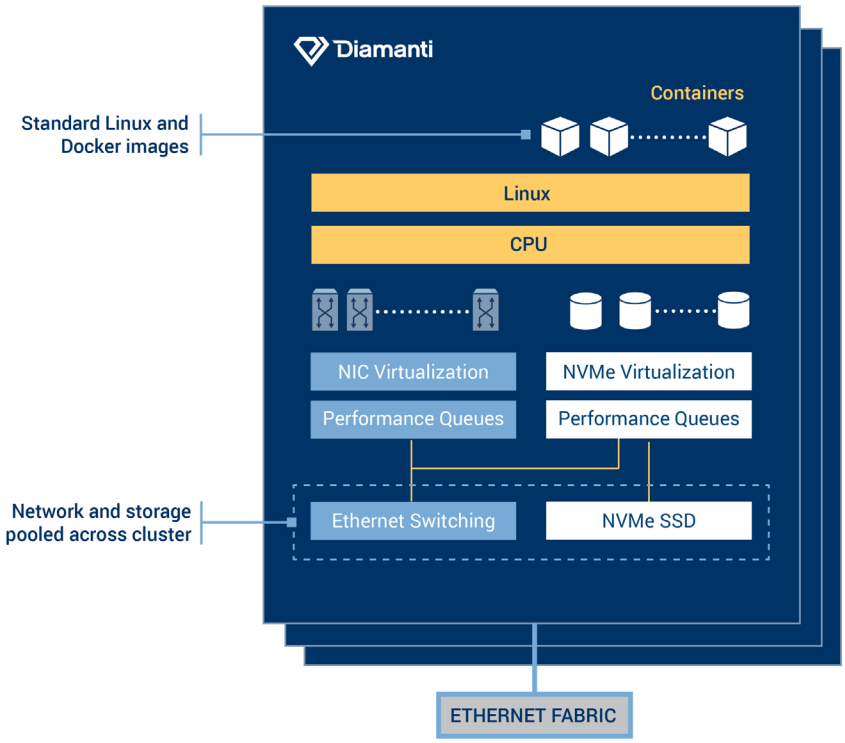


Figure 1. Diamanti D-Series Architecture

Diamanti D-Series appliances assure network performance by enforcing performance tiers at the network interface for each container. Each container is guaranteed a share of available network bandwidth. Similarly, D-Series appliances ensure storage performance through bandwidth, IOPS, and latency guarantees. These guarantees are enforced in hardware, solving issues related to noisy neighbors in multitenancy environments.

Introducing Diamanti D-Series Clusters

The Diamanti software aggregates resources from individual member nodes and groups the nodes into a cluster. After a Diamanti cluster is formed, a virtual IP address (VIP) is assigned to the cluster, which provides access to a single point of management.

Note: This virtual IP address is also referred to as the management IP address for the cluster in the *Diamanti* documentation.

Administrators have access to both a command line interface and an easy-to-use browser-based user interface to manage and monitor the cluster.

Note: The Diamanti D-Series cluster is built with a typical quorum size of three nodes.

Exploring Container Networking

The Diamanti software provides centralized management of network configurations, which are then realized across a distributed cluster. Diamanti addresses application networking needs to meet both the connectivity and service requirements of the management network and the performance requirements of applications using data networks.

The Diamanti software does this by providing capabilities to attach multiple interfaces to a single container. This approach further allows network administrators to maintain isolation between management and data networks.

Container Management Network

By default, the Diamanti software provides a management network for containers allowing service discovery. This management network is a static subnet (172.20.0.0/24) on each node.

Container Data Networks

Each container can request one or more interfaces on specified data networks. Further, each data interface can be programmed to have a specific performance configuration. The Diamanti software provisions the data network requirements using SR-IOV NICs on the nodes.

For data networking, the Diamanti software employs two related objects: networks and endpoints.

Networks

Diamanti network objects comprise IP address pools from which containers can have IP addresses allocated. The Diamanti software provides layer 2 (L2) networking with VLANs.

Diamanti network objects allow network administrators to configure the following parameters:

- Network name, which is a user-defined string that identifies the network object
- Subnet for the IP address range
- IP address range
- Default gateway for the subnet (optional)
- VLAN corresponding to the subnet (which is programmed on the Top-of-Rack switch)

These parameters allow network administrators to address most network topologies, including the following:

- Public networks—Required if containers need to communicate with external components on a different subnet. In this case, you can configure the gateway, as needed.
- Private networks—Available if containers do not need to communicate with external components, or are running in a restricted environment. In this case, network administrators should not configure the gateway.

Note that private networks allow users to reuse the same subnet in multiple clusters without conflict as long as the ToR switches are appropriately configured for the VLAN.

Endpoints

Container networking is provisioned through endpoints. Endpoints associate container networking requirements with network objects and performance guarantees. Endpoints are allocated their IP address from the specified networks. Each endpoint uses a dedicated SR-IOV virtual function (VNIC) to provide networking for the container.

Endpoints can be either persistent (named) or dynamic. Users need to explicitly create or delete persistent endpoints. Because of this, persistent endpoints are suitable for services that require a fixed IP address. Dynamic endpoints, in contrast, are created when a pod is created, and deleted automatically when the pod is deleted.

Note: MAC addresses are automatically assigned without the need for user configuration.

Exploring Storage

The Diamanti software includes an advanced distributed volume manager that pools storage from drives available on all appliances in the cluster.

Note: Each volume uses dedicated SR-IOV virtual function to provide storage for the container.

Diamanti Converged File System

Diamanti Converged File System (DCFS) is the distributed storage component of the Diamanti cluster. DCFS consists of a highly-available and distributed control plane that scales linearly when the number of nodes in the cluster increases.

DCFS provides application-centric storage features that include mirroring, remote I/O, snapshots, and more. DCFS is network aware and automatically reserves network bandwidth as needed to maintain application quality of service. DCFS is flash media aware and designed for higher flash endurance and predictable I/O latency.

Volumes

Volumes of various sizes can be created from the distributed storage pool on demand. When a volume is created, the Diamanti scheduler automatically picks an optimum node on which to create the volume.

Remote Volumes

Volumes created on a node are accessible by other nodes using the Ethernet fabric. These remotely-accessed volumes are referred to as *remote volumes*.

Note that the Diamanti scheduler prefers local volumes and attempts to place containers on the nodes where the volumes are located.

Mirrored Volumes

Volumes can be created to have mirrored copies on nodes (up to three copies). This ensures that there is no loss of data if a node or drive fails.

Snapshots

A snapshot represents a point-in-time image of the corresponding volume data. Diamanti snapshots are read-only. Users need to create a volume from a snapshot to consume snapshot data.

Linked Clones

Volumes created from Diamanti snapshots are called linked clones of the parent volume. Linked clones share data blocks with the corresponding snapshot until the linked clone blocks are modified. Linked clones can be attached to any node in the cluster for consumption.

Dynamic Provisioning

The Diamanti software offers a dynamic provisioner for Kubernetes, allowing storage administrators to dynamically provision storage volumes. Dynamic provisioning uses the concept of storage classes, which enables storage administrators to define classes of storage for use with varying application requirements.

Understanding Namespaces

The Diamanti software supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. Namespaces are helpful in environments with many users spread across multiple teams or projects. In this case, namespaces provide a private workspace for each team or deployment.

Namespaces do this by enforcing a scope for names. When using namespaces, names of resources need to be unique within a namespace, but not across namespaces. In this way, namespaces allow cluster administrators to divide cluster resources between multiple uses.

Understanding Performance Tiers

The Diamanti software allows users to run applications using different performance tiers based on specific service-level agreement (SLA) requirements. By default, the Diamanti software offers three predefined performance tiers. Cluster administrators can add new performance tiers, up to a total of eight, based on user requirements.

User Access Management

The Diamanti software provides built-in role-based access control (RBAC) to regulate access to resources within the environment, providing a streamlined and secure mechanism to perform cluster, container, and user administration. The Diamanti software offers several built-in roles and groups to cover the most common administrative tasks. Cluster administrators can also create custom groups to match specific needs, as well as define users within the cluster.

The Diamanti software also supports the namespaces construct in Kubernetes to manage access to application and service-related Kubernetes objects. These objects include pods, services, replication controllers, deployments, and other objects that are created in namespaces.

Diamanti roles are expressed using the following format:

```
<class>-<qualifier>[/<namespace>]
```

where `<class>` refers to the object type (such as container, network, or node, among others), `<qualifier>` specifies the permitted operations (view or edit), and `<namespace>` identifies the namespace to which the role applies.

For example, built-in roles include `network-edit` and `container-edit/project`, among others.

Note: Cluster administrators can configure additional RBAC privileges for Kubernetes objects using Kubernetes role and role bindings. Diamanti built-in roles and bindings are identified using the label `diamanti.com/created-by`. Custom roles and bindings must use a different label.

Kubernetes

The D-Series appliance is packaged with a certified version of Kubernetes (version 1.12.3). The Diamanti network CNI plug-in and the storage plug-in (via Flex-volume/CSI) are pre-integrated and tested with the Kubernetes distribution.

Note: Do not upgrade the Kubernetes software independently. The Kubernetes software is upgraded, if necessary, as part of the Diamanti software upgrade process.

For more information on Kubernetes and supported features, refer to the Kubernetes documentation.

2

Managing the D-Series Appliance

This chapter describes how to manage the Diamanti D-Series appliance. The chapter begins with an overview of how to deploy the Diamanti D-Series appliance, and then provides detailed instructions about how to cluster multiple D-Series appliances.

The chapter further explains how to monitor and manage the cluster, as well as how to manage groups of users. The chapter then provides information about managing key Diamanti objects including performance tiers, networks, endpoints, and storage. The chapter also describes how to monitor the health of the D-Series appliance.

The chapter concludes by showing how to deploy workloads as containers using Kubernetes pod definitions, and how to use Kubernetes commands to run, monitor and delete pods.

Deploying the D-Series Appliance

Deploying the Diamanti D-Series appliance is a quick and simple process that involves the following steps:

- Becoming familiar with the Diamanti D-Series package contents and collecting the necessary hardware and configuration information
- Physically installing the D-Series appliance in the data center
- Configuring the D-Series appliance to prepare it to join a cluster

For more information about deploying a Diamanti D-Series appliance, refer to the *Diamanti D-Series Installation Guide*.

Clustering D-Series Appliances

Cluster administrators can create a cluster of D-Series appliances using the Diamanti software, and add or remove nodes in the cluster as needed. After a cluster is formed, the Diamanti software pools resources across all nodes in the cluster, enabling Kubernetes to efficiently schedule containers within the cluster.

Important: Complete the *Diamanti Cluster Preparation Checklist*, available in the *Diamanti D-Series Installation Guide*, before configuring a cluster.

Cluster administrators interact with the Diamanti software using the Diamanti command line interface (CLI), which provides a set of commands to manage Diamanti D-Series clusters, nodes (appliances), storage volumes, networks, and more. In contrast, users employ standard Kubernetes commands to manage pods, containers, and other Kubernetes resources in the cluster.

Use the Diamanti command line tool `dctl` (Linux and Mac OS X) and `dctl.exe` (Windows) to issue Diamanti commands, and use the Kubernetes command line tool `kubectl` (Linux and Mac OS X) and `kubectl.exe` (Windows) to issue Kubernetes commands.

Note: Diamanti refers to these tools as `dctl` and `kubectl` respectively throughout the documentation.

Diamanti strongly recommends running these tools on a local Linux, Windows, or Mac OS X workstation. Users can download the Diamanti tools to run the Diamanti CLI on a local machine. For more information about downloading the tools, refer to the *Diamanti D-Series User Interface Guide*.

Preparing to Cluster Appliances

Before creating a cluster, ensure that the host names and cluster name have DNS entries for the corresponding domain. If these DNS entries are not present (and therefore cannot be resolved by nodes), attempts to create the cluster will fail.

To verify that the host names and cluster name have DNS entries, do the following:

1. Using a console monitor, log in to the node.
2. Confirm that nameserver entries are correctly configured using the following command:

   ```
   $ cat /etc/resolv.conf
   ```

   The `resolv.conf` file typically contains directives that specify the default search domains along with the list of IP addresses of nameservers available for resolution.
3. Perform a DNS query to test the address mapping of a well-known site. For example:

   ```
   $ nslookup google.com
   ```

Creating a Cluster

To create a secure cluster, cluster administrators need to specify the virtual IP address for the cluster, the DNS sub-domain, the VLAN used for intra-cluster storage communication, an admin user password for the cluster, the certificate authority for the cluster, and the TLS certificate and private key for the cluster.

Note: All commands, except the `dctl cluster create` command, require administrators to be logged into the cluster (using the `dctl login` command).

Use the following command:

```
dctl -c <path-to-config-file> -s <node-name | node-ip-address>
   cluster create <cluster-name>
   <dns-name>,[<dns-name>,...]
   --vip <cluster-virtual-ip>
   --poddns <cluster-name>[.company.domain]
   --storage-vlan [vlan-id]
   --admin-password <password>
   --ca-cert value <path-to-certificate>
   --tls-cert value <path-to-certificate>
   --tls-key value <path-to-key-file>
```

The following describes the command parameters:

Parameter	Description
--admin-password <password> -p <password>	The password for the admin user. If a password is not specified in the command line, the command prompts for a password. The password needs to meet the Unicode standard, and must consist of at least 8 characters, including at least one special character, one upper and lowercase character, and one number.
-c <path-to-config-file>	The path to the cluster creation configuration file containing the cluster name, admin user password, certificate authority, server private key, storage VLAN, virtual IP address, and list of nodes. See Appendix A for an example of a configuration file.
--ca-cert value <path-to-certificate>	The certificate authority for the cluster.
<cluster-name>	The name of the cluster.
<dns-name>	The DNS (short) name of the node.
--poddns <cluster-name>[.company .domain]	The domain name for the DNS service for all containers deployed in the cluster. The default is `domain.com`.
-s <node-name | node-ip-address>	The DNS (short) name or IP address of the Diamanti server that is a member of the cluster. Use when running the command from a Linux, Windows, or Mac-based machine. The command defaults to 127.0.0.1 if the node IP address is not specified. Therefore, the node IP address is not required when the command is run on a node that is part of the cluster.
--storage-vlan [vlan-id] --svlan [vlan-id]	The virtual LAN ID (for intra-cluster storage communication, with jumbo frames enabled). The default value is 0 (disabled).
--tls-cert value <path-to-certificate>	The TLS certificate for the cluster.
--tls-key value <path-to-key-file>	The TLS private key for the cluster.
--vip <cluster-virtual-ip> --virtual-ip <cluster-virtual-ip>	The IP address for managing the cluster.

For example:

```
$ dctl cluster create cluster appserv76,appserv77,appserv78 --vip 172.16.20.251
--poddns cluster.local --svlan 500 -p Test1234! --ca-cert /home/diamanti/cluster/ca.crt
--tls-cert /home/diamanti/cluster/cluster.crt --tls-key /home/diamanti/cluster/cluster.key
Name            : cluster
UUID            : 9cb6e1b4-5f3d-11e8-9adc-a4bf01147a45
State           : Creating
Master          :
Etcd State      :
Virtual IP      : 172.16.20.251
Storage VLAN    : 500
Pod DNS Domain  : cluster.local

NAME        NODE-STATUS   K8S-STATUS   MILLICORES   MEMORY    STORAGE   IOPS      VNICS
BANDWIDTH   SCTRLS          LABELS

LOCAL, REMOTE
appserv76                          0/0         0/0      0/0       0/0       0/0       0/0
0/0, 0/0        <none>
appserv77                          0/0         0/0      0/0       0/0       0/0       0/0
0/0, 0/0        <none>
appserv78                          0/0         0/0      0/0       0/0       0/0       0/0
0/0, 0/0        <none>
```

Important: Do not add spaces or other whitespace characters when specifying the list of nodes (using DNS short names).

Note that the `dctl cluster create` command automatically adds an administrative user named 'admin'. Using the default quorum size of three nodes, the first three nodes specified in the `dctl cluster create` command become quorum nodes by default.

Logging in to the Cluster

Users need to be logged in to the cluster before performing CLI operations. There are two ways to log in to a cluster:

- Insecure mode—Use when user-supplied certificates were used to create the cluster and the certificate authority (ca.crt) is not available for the user. This allows the user to configure `dctl` to ignore server certificate validation. This is the default login mode.
- Kubernetes proxy mode—In certain environments (such as with an SSL passthrough load-balancer), `kubectl` commands are not properly relayed to the API server causing `kubectl` commands to fail with an error message. In these cases, use the proxy option, which relays `kubectl` commands through the Diamanti server.

Use the following command to log in to the Diamanti cluster:

```
dctl [-s <cluster-vip | cluster-dns-name>] login
```

Users are prompted for the user name and the password. For example:

```
$ dctl -s 172.16.19.57 login
Username: admin
Password:
```

After successfully logging in to the cluster, the `dctl login` command creates a D-Series cluster session cookie in the `.dctl.d` directory. Note that this cookie is deleted when users log out using `dctl logout` command.

The `dctl login` command also creates a `.kube/config` file that contains the Kubernetes cookie together with the context of the D-Series cluster. Users can interact with the Kubernetes API Server after this information is available.

Users can determine the identity of the user currently logged in using the following command:

```
dctl whoami
```

For example:

```
$ dctl whoami
Name:          admin
Built-In:      true
Local-Auth:    true
Groups:        user-admin, cluster-admin
Roles:         node-edit, perftier-edit, volume-edit, required, user-edit,
allcontainer-edit, volumeclaim-edit/default, network-edit, container-edit/
default
Namespace:     default
```

To log out of the cluster, use the following command:

```
$ dctl logout
```

CLI Configuration Management

After successfully logging in to the cluster, the `dctl login` command creates D-Series cluster session cookies, which are deleted when users log out using the `dctl logout` command. The `dctl login` command also creates Kubernetes session tokens that contain the Kubernetes cookie together with the context of the D-Series cluster.

These session tokens are stored under the home directory in the following files:

File	Description
~/.dctl.d/cookies	Diamanti session cookies
~/.kube/config	Kubernetes session tokens

Note: The user session tokens expire in one hour.

Users can specify alternative locations for these configuration files using the following commands:

```
export DCTL_CONFIG=<path>
export KUBECONFIG=${DCTL_CONFIG}/.dctl.d/kubeconfig
```

For example:

```
$ export DCTL_CONFIG=/home/johnsmith
$ export KUBECONFIG=${DCTL_CONFIG}/.dctl.d/kubeconfig
```

Note: If you export `DCTL_CONFIG`, you need to also export `KUBECONFIG` as shown above.

When destroying the cluster, users should remove the corresponding folders on the machine before using the Diamanti CLI (`dctl` commands) again.

Monitoring and Managing the Cluster

After creating the cluster and logging in to it, users can monitor the cluster status, add more nodes to the cluster, label nodes, and more.

Use the following command to display the status of the cluster:

```
dctl cluster status
```

For example:

```
$ dctl cluster status
Name            : production-cluster
UUID            : efbe3403-661d-11e9-aff6-2c600c82ec99
State           : Created
Version         : 2.3.0 (108)
Master          : appserv3
Etcd State      : Healthy
Virtual IP      : 172.16.19.21
Storage VLAN    : 402
Pod DNS Domain  : cluster.local
NAME                    NODE-STATUS   K8S-STATUS   MILLICORES   MEMORY          STORAGE
IOPS        VNICS     BANDWIDTH   SCTRLS          LABELS

LOCAL, REMOTE
appserv2/(etcd)          Failed       Good        0/32000      1GiB/64GiB      200.54GB/3.05TB
185K/500K   20/63     4.63G/40G   20/64, 0/64     beta.kubernetes.
io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=appserv2
appserv3/(master, etcd)   Good         Good        250/32000    1.06GiB/64GiB   258.36GB/3.05TB
185K/500K   25/63     4.63G/40G   21/64, 0/64     beta.kubernetes.
io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=appserv3
appserv4/(etcd)          Good         Good        0/32000      1GiB/64GiB      1.07TB/3.05TB
185K/500K   24/63     4.63G/40G   20/64, 0/64     beta.kubernetes.
io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=appserv4
```

Use the following command to add one or more nodes to a cluster:

```
dctl cluster add <dns-name>,[<dns-name>,...]
```

For example:

```
$ dctl cluster add host1,host2
```

Important: Do not add spaces or other whitespace characters when specifying the list of nodes (using DNS short names).

Similarly, use the following command to remove one or more nodes from a cluster:

```
dctl cluster remove <dns-name>,[<dns-name>,...]
```

For example:

```
$ dctl cluster remove host1
```

You are prompted for confirmation.

Use the following command to add a label to a node:

```
dctl node label <node-name> <label>
```

For example:

```
$ dctl node label appserv1 node=web-tier
Node update succeeded
```

Managing User Groups

The Diamanti software uses role-based access control (RBAC) to regulate access to resources within the environment. As part of this system, Diamanti D-Series appliances employ the following related concepts:

- Users—People who can perform tasks within the environment.
- Roles—Collections of functions that users can perform. Within the Diamanti software, these functions are defined as methods that can act on objects within the system. There is a fixed set of predefined roles that cannot be changed.
- Groups—Collections of users that assume the same role within the system. User groups define the privileges that users are assigned in the system.

Note: Users can belong to multiple groups, and groups can have multiple roles.

The Diamanti software offers built-in roles and groups to cover the most common administrative tasks. Cluster administrators can also create custom groups to match specific needs, as well as define users within the cluster.

Object	Name	Description
User	admin	Belongs to the cluster-admin group.
Group	cluster-admin	Has edit and view privileges for Diamanti resources including containers, volume claims, networks, nodes, performance tiers, and volumes. Note that this group offers full privileges in Kubernetes.
	container-admin	Has edit privileges for containers in the default namespace and view privileges for Diamanti resources including networks, nodes, performance tiers, and volumes.
	user-admin	Has view and edit privileges for users, group, and authentication servers.

Cluster administrators generally perform the following tasks:

- Manage the cluster configuration
- Monitor container deployments
- Monitor hosts, volumes, and networks
- Monitor and manage common operations of the IT infrastructure, including Diamanti D-Series appliances

Use the following command to display the list of roles associated with the user:

```
$ dctl user role list
NAME                BUILT-IN   SCOPE
allcontainer-edit   true       Cluster
allcontainer-view   true       Cluster
container-edit      true       Resource
container-view      true       Resource
network-edit        true       Cluster
network-view        true       Cluster
node-edit           true       Cluster
node-view           true       Cluster
perftier-edit       true       Cluster
perftier-view       true       Cluster
required            true       Cluster
user-edit           true       Cluster
user-view           true       Cluster
volume-edit         true       Cluster
volume-view         true       Cluster
volumeclaim-edit    true       Resource
```

Use the following command to create a new group with role based access control:

```
$ dctl user group create mygroup --role-list user-view
NAME      BUILT-IN   ROLE LIST             EXTERNAL GROUP
mygroup   false      required, user-view
```

Use the following command to get details about a particular group:

```
$ dctl user group get cluster-admin
NAME     BUILT-IN ROLE LIST
EXTERNAL GROUP
cluster-admin   true       allcontainer-edit, container-edit/default,
network-edit, node-edit, perftier-edit, required, volume-edit, volumeclaim-
edit/default
```

Use the following command to create a new user and add the user to multiple user groups:

```
$ dctl user create myuser@adexample.com --group-list mygroup,cluster-admin
Name:         myuser@adexample.com
Built-In:     false
Local-Auth:   false
Groups:       mygroup, cluster-admin
Roles:        user-view, required, network-edit, volume-edit, allcontainer-
edit, container-edit/default, volumeclaim-edit/default, node-edit, perftier-
edit
Namespace:    default
```

Note: User names need to be in the following format: `user@domain`.

Performing Authentication

Local users can be authenticated, as shown in the following example:

```
$ dctl login
Name        : cluster6
Virtual IP: 172.16.19.23
Server      : cluster6.eng.diamanti.com
WARNING: Thumbprint : 51 57 1f 38 c4 9a f4 24 50 1c 9b 3c 91 f5 07 25 96 c9
6c 18 b5 55 b4 1f 4f f9 f4 2c a4 26 e7 88
[CN:diamanti-signer@1557946544, OU:[], O=[] issued by CN:diamanti-
signer@1557946544, OU:[], O=[]]
Username: myuser2
Password:
Successfully logged in

$ dctl whoami
Name:         myuser2
Built-In:     false
Local-Auth:   true
Groups:       container-admin, node-edit
Roles:        volume-view, required, container-edit/default, network-view,
node-edit, perftier-view
Namespace:    default
```

```
$ dctl passwd
Current password:
New password:
Confirm password:
Name:         myuser2
Built-In:     false
Local-Auth:   true
Groups:       container-admin, node-edit
Roles:        node-edit, perftier-view, volume-view, required, container-
edit/default, network-view
Namespace:    default

$ dctl logout
```

Using LDAP Authentication

Users can be authenticated through an external LDAP server. The following shows an example of remote user authentication:

```
$ dctl user auth-server create myauth --server ldap.example.com --port 389
--base-dn 'dc=example,dc=com' --bind-dn 'cn=mybinddn'
--bind-password 'mybindpassword' --user-filter '(sAMAccountName=%s)'

Name        : myauth
Server      : ldap.example.com
Port        : 389
LDAP Base   : dc=example,dc=com
Reachable   : true

$ dctl user create myremoteuser@adexample.com --group-list cluster-admin
Name:         myremoteuser@adexample.com
Built-In:     false
Local-Auth:   false
Groups:       cluster-admin
Roles:        required, allcontainer-edit, container-edit/default,
volumeclaim-edit/default, network-edit, node-edit, perftier-edit, volume-
edit
Namespace:    default
```

```
$ dctl login
Name        : devtb6
Virtual IP: 172.16.19.23
Server      : devtb6.eng.diamanti.com
WARNING: Thumbprint : 51 57 1f 38 c4 9a f4 24 50 1c 9b 3c 91 f5 07 25 96 c9
6c 18 b5 55 b4 1f 4f f9 f4 2c a4 26 e7 88
[CN:diamanti-signer@1557946544, OU:[], O=[] issued by CN:diamanti-
signer@1557946544, OU:[], O=[]]
Username: myremoteuser@adexample.com
Password:
Successfully logged in

$ dctl whoami
Name:         myremoteuser@adexample.com
Built-In:     false
Local-Auth:   false
Groups:       cluster-admin
Roles:        volumeclaim-edit/default, network-edit, node-edit, perftier-
edit, volume-edit, required, allcontainer-edit, container-edit/default
Namespace:    default
```

Managing Performance Tiers

Administrators can configure the Diamanti software to enforce minimum network throughput and storage IOPS for containers, offering deterministic high performance with high workload density. While the configuration is completed globally across the cluster, each node in the cluster enforces the specified performance tier for all workloads running on the node.

The following table outlines the three built-in performance tiers in a Diamanti cluster:

Performance Tier	Storage IOPS	Network Bandwidth
high	20K IOPS	500 Mbps
medium	5K IOPS	125 Mbps
best-effort	No minimum	No minimum

Administrators can create new performance tiers to match application requirements using the Diamanti CLI. Administrators can define five additional performance tiers (for a total of eight tiers). Administrators cannot modify or delete the default built-in performance tier “best-effort.” If a performance tier is not specified, the default is “best-effort.”

Note: Diamanti performance tiers represent guaranteed minimum performance. Performance maximums are not enforced. Higher performance workloads are prioritized over “best-effort” workloads.

Use the following command to create a performance tier:

```
dctl perf-tier create <performance-tier-name> -i <storage-iops>
-b <network-bandwidth>
```

For example:

```
$ dctl perf-tier create performance-tier1 -i 1K -b 1G
```

Use the following command to delete a performance tier:

```
dctl perf-tier delete <performance-tier-name>
```

Note: You cannot delete the “best-effort” performance tier.

Managing Networks

The Diamanti software enables you to manage networks as objects. A network object consists of an IP address range, the subnet, the VLAN associated with the subnet, and the default gateway.

Containers can connect to multiple networks using endpoints. The endpoint specifies the network object and the performance tier to be used by the container.

Use the following command to create a network:

```
dctl network create <network-name> --subnet <subnet/class>
   --start <ip-range-start> --end <ip-range-end>
   --gateway <gateway> --vlan <vlan-id>
```

The following table describes the command options:

Parameter	Description
<network-name>	The name of the network.
--subnet <subnet/class>	The subnet for the network. The command defaults to the subnet you have already specified, though you can change the value, as required.
--start <ip-range-start>	The starting address of the range of IP addresses to allocate to containers on the network.
--end <ip-range-end>	The final address of the range of IP addresses to allocate to containers on the network.

Parameter	Description
--gateway <gateway>	The IP address of the gateway. The command defaults to the gateway you have already specified, though you can change the value, as required.
--vlan <vlan-id>	The ID of the virtual LAN on which the subnet should be placed based on your network configuration.

For example:

```
$ dctl network create northbound --subnet 192.168.30.0/24
--start 192.168.30.101 --end 192.168.30.200 -g 192.168.30.1 -v 1004
```

You can verify that the network was successfully created with the attributes you specified using the following command:

```
$ dctl network list
```

After creating a network, you can include the network by name in your pod definition files. The following highlights the relevant section of the pod definition file containing the network specification:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    diamanti.com/endpoint0: '{"name":"0","network":"northbound","perfTier":"medium"}'
  name: hello
spec:
  containers:
  - image: docker.io/nginx
    imagePullPolicy: Always
    name: hello
    resources:
      limits:
        cpu: 500m
        memory: 64Mi
      requests:
        cpu: 500m
        memory: 64Mi
```

```
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: volume-1
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
  - flexVolume:
      driver: diamanti.com/volume
      fsType: ext4
      options:
        name: nginx-vol
        perfTier: medium
    name: volume-1
```

The example shows an optional performance tier assigned to the interface to enforce guaranteed performance.

Note: The Diamanti software requires that all performance tier settings within a single pod definition file be identical. In other words, all network interfaces and storage volumes should be set to the same performance tier. If a performance tier is not specified, the Diamanti software assumes "best-effort."

To delete a network from your Diamanti cluster, use the following command:

```
dctl network delete <network-name>
```

Note: You can only delete networks that are not being used. You are prompted to confirm the delete operation.

For example, the following command removes the "northbound" network that you created earlier:

```
$ dctl network delete northbound
```

Managing Endpoints

The Diamanti software enables you to provision container networking through endpoints that associate container networking requirements with network objects and performance tiers. Endpoints can be either dynamic or persistent.

Dynamic endpoints are created automatically when a pod is scheduled, and are deleted when the pod terminates. These endpoints are consumed by a single pod/service, and are specified inside the pod definition file, as shown in the following:

```
...
annotations:
  diamanti.com/endpoint0: '{"network":"blue","perfTier":"high"}'
...
```

Persistent endpoints are named and can have either an IP address assigned automatically or have an IP address explicitly assigned. Use the following command to create a persistent endpoint, automatically assign an IP address, and assign the endpoint to a namespace:

```
dctl endpoint create <endpoint-name> -n <network-name> -ns <namespace>
```

For example:

```
$ dctl endpoint create ep1 -n blue -ns default
NAME   NAMESPACE   CONTAINER  NETWORK   IP               MAC   GATEWAY            VLAN   VF
PORT      NODE       LABELS
ep1    default                blue      172.16.155.4/24        172.16.155.1 155
                     <none>
```

Alternatively, use the following command to create a persistent endpoint and explicitly assign an IP address:

```
dctl endpoint create <endpoint-name> -n <network-name> -i <ip-address>
```

For example:

```
$ dctl endpoint create ep1 -n blue -i 172.16.155.4
NAME   NAMESPACE   CONTAINER  NETWORK   IP               MAC   GATEWAY            VLAN   VF
PORT      NODE       LABELS
ep1    default                blue      172.16.155.4/24        172.16.155.1 155
                     <none>
```

Persistent endpoints are similarly consumed by a single pod/service, and are specified inside the pod definition file, as shown in the following:

```
...
annotations:
  "diamanti.com/endpoint0": '{\"network\":\"blue\",\"perfTier\":\"high\"}'
  "diamanti.com/endpoint1": '{\"network\":\"red\",\"perfTier\":\"low\"}'
...
```

To list the defined endpoints, use the following command:

```
dctl endpoint list
```

To show detailed information about a specific endpoint, use the following command:

```
dctl endpoint get <endpoint-name>
```

Use the following command to delete a persistent endpoint:

```
dctl endpoint delete <endpoint-name>
```

For example:

```
$ dctl endpoint delete mysqlep
```

Note: Each pod has a management network interface and, optionally, multiple data network endpoints. The management network provides connectivity from pods to nodes, the Kubernetes service, DNS, and more. Management network provisioning is automatic, but you are responsible for specifying the data endpoints.

Managing Storage

The Diamanti software enables you to create and attach multiple storage volumes to a container and optionally guarantee minimum throughput using selected performance tiers.

Use the following command to create a storage volume:

```
dctl volume create <volume-name> -s <size>
```

For example:

```
$ dctl volume create mongo-vol -s 20G
```

This creates a new volume called `mongo-vol` allocating 20GB of storage.

After creating the storage volume, you need to modify the pod definition file to instruct the appropriate container to use the volume. For example, you could use the volume created in this section by adding the text in bold below:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    diamanti.com/endpoint0: '{"name":"0","network":"northbound","perfTier":"medium"}'
  name: mongo-app
spec:
  containers:
  - image: docker.io/mongo
    imagePullPolicy: Always
    name: mongo-app
    resources:
      limits:
        cpu: 500m
        memory: 64Mi
      requests:
        cpu: 500m
        memory: 64Mi
```

```
    volumeMounts:
    - mountPath: /usr/share/mongo/html
      name: mongo-vol
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:
  - flexVolume:
      driver: diamanti.com/volume
      fsType: ext4
      options:
        name: mongo-vol
        perfTier: medium
        detachPolicy: auto
    name: mongo-vol
```

The example shows an optional performance tier assigned to the volume to enforce guaranteed performance. You can also specify a placement constraint using the `--sel=<label>` option to create a volume on a specific node or on a node with specific characteristics.

Note: The Diamanti software requires that all performance tier settings within a single pod definition file be identical. In other words, all network interfaces and storage volumes should be set to the same performance tier. If a performance tier is not specified, the Diamanti software assumes "best-effort."

The previous example also shows how to configure the detach policy for a volume using the `detachPolicy` option. The detach policy specifies whether a mirrored or remote volume can be detached automatically when an initiator node is not reachable. Note that detaching a volume automatically can lead to data loss under certain network partitioning conditions; use this option with care.

The following settings are available for the `detachPolicy` option:

Setting	Description
Auto	(Default) The volume is force detached automatically when the initiator node is not reachable and the corresponding pod is evicted. This enables the workload to be moved to a different node automatically without requiring user intervention to force detach volumes. Note that pods are evicted after five minutes when a node is not reachable or fails.

Setting	Description
Manual	This is the default setting, in which volumes are not detached automatically when an initiator node is unreachable (even when the corresponding pod is evicted). Use this setting when the combination of pod eviction and a node being unreachable does not imply that the corresponding pods have to stop running immediately. In this case, since the pod might still be using the volume, an automatic force detach on the initiator could lead to data loss. Note that you can always override this setting and manually force detach volumes, as needed.

PersistentVolume and PersistentVolumeClaim

The Diamanti software supports PersistentVolume (PV) and PersistentVolumeClaim (PVC). A PersistentVolume is storage that has been provisioned by an administrator and has a lifecycle independent of any individual pod that uses the PV.

A PersistentVolumeClaim (PVC) is a request for storage by a user. PVCs consume PV resources, with a claim requesting a specific size and access mode. Volumes persist across container failures and restarts.

Each PV contains a spec and status, which is the specification and status of the volume. For example:

```
apiVersion: v1
items:
- apiVersion: v1
  kind: PersistentVolume
  metadata:
    creationTimestamp: null
    labels:
      volume: mongo-1
    name: mongo-1-pv
  spec:
    accessModes:
    - ReadWriteOnce
    capacity:
      storage: 100Gi
    flexVolume:
      driver: diamanti.com/volume
      fsType: xfs
      options:
        name: mongo-test-1
        perfTier: high
    persistentVolumeReclaimPolicy: Retain
```

Similarly, each PVC contains a spec and status, which is the specification and status of the claim. For example:

```
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    creationTimestamp: null
    name: mongo-1-claim
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 100Gi
    selector:
      matchLabels:
        volume: mongo-1
```

The following example shows how to specify a PVC in a pod definition file:

```
...
          volumeMounts:
          - mountPath: /data/db
            name: mongo-vol
        dnsPolicy: ClusterFirst
        nodeSelector:
          kubernetes.io/hostname: diamanti-2
        restartPolicy: Always
        securityContext: {}
        terminationGracePeriodSeconds: 30
        volumes:
        - name: mongo-vol
          persistentVolumeClaim:
            claimName: mongo-1-claim
```

Note: The `volumeMounts name` value must be the same as the `volumes name` setting. Also, Kubernetes requires that the `volumeMounts name` value must be unique within a node (and preferably across the cluster).

Dynamic Provisioning

You can specify the dynamic provisioning of volumes, in which volumes are dynamically created and optionally deleted on request. To use dynamic provisioning, you begin by creating a storage class that specifies the name, file system type, performance tier, mirror count, and provisioner, as shown in the following example:

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: storageclass-high
provisioner: diamanti.com/default-provisioner
parameters:
  perfTier: high
  fsType: xfs
  mirrorCount: "3"
  driver: diamanti.com/volume
```

In this case, the provisioner is `diamanti.com/default-provisioner.`

The dynamic provisioner uses the storage classes you have defined to create requested volumes dynamically when a user creates a PVC (specifying the volume size and storage class), as shown in the following:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pv1
  annotations:
    volume.beta.kubernetes.io/storage-class: storageclass-high
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Mirroring

The Diamanti software additionally supports mirroring (up to three-way mirrors). Each mirror or plex exists on a single node. The Diamanti scheduler auto selects and spreads the mirrors on different nodes in the cluster. You can also optionally override this auto-selection using a selector.

Use the following command to create a mirrored volume:

```
dctl volume create <volume-name> -s <size> -m <mirror-count>
```

For example:

```
$ dctl volume create vol1 -s 100G -m 3
NAME      SIZE      NODE      LABELS    PHASE     ATTACH-STATUS   ATTACHED-TO   DEVICE-PATH   AGE
voln      100G      []        <none>    Pending                                                 0s
```

This creates a three-way mirrored volume.

Use the following command to display the number of nodes for a volume:

```
dctl volume describe <volume-name>
```

For example:

```
$ dctl volume describe vol1
Name                    : vol1
Size                    : 20.03GiB
Node                    : [appserv1 appserv2 appserv3]
Label                   : <none>
Phase                   : Available
Attached-To             :
Device Path             :
Age                     : 10m

Plexes:
         NAME       NODES          STATE     ATTACH-STATE   CONDITION   OUT-OF-SYNC-AGE
          -----      ------------   ------    ------------   ---------   ------------
          vol1.p0    appserv1       Up        Detached       InSync
          vol1.p1    appserv2       Up        Detached       Detached
          vol1.p2    appserv3       Up        Detached       Detached
```

You can add another mirror to volumes with one or two mirrors (one mirror at a time) using the following command:

```
dctl volume modify <volume-name> -m <new-mirror-count>
```

Similarly, you can delete a mirror (plex) associated with a volume using the following command:

```
dctl volume plex-delete <volume-name> p<plex-number>
```

For example:

```
dctl volume plex-delete vol1 p1
```

You can delete storage volumes that are no longer needed. Recall that persistent volumes persist across container failures and restarts.

Use the following command to delete a storage volume:

```
dctl volume delete <volume-name>
```

For example:

```
$ dctl volume delete mongo-vol
```

Note: You cannot delete a volume that is in use. Also, deleting a volume is an irrevocable action. Exercise care when deleting a storage volume.

Diamanti software periodically monitors all volumes and checks for active volumes in degraded state. Degraded state is an indication that one or more plexes in the volume are in `Detached` condition. If any are found, Diamanti software tries to auto attach the plex if the node on which the plex resides and corresponding storage links are in good condition.

The auto-attach frequency is configured in the `/etc/diamanti/convoy.conf` file using the following parameter:

```
DIAMANTI_PLEX_AUTO_ATTACH_INTERVAL
```

The default value is 30 minutes. Diamanti software attempts to attach the detached plexes three times within this configured interval. If the plex attach is not successful, it is scheduled again for after the configured interval. Note that changes to this interval requires that you restart the `convoy` service on the master node.

Monitoring Health

The Diamanti D-Series appliance is a sophisticated platform with multiple hardware and software components. Administrators need to regularly monitor the health of D-Series nodes in the cluster. If a node is unhealthy, new pods are not scheduled to run on the node.

The Diamanti software automatically corrects most transient issues and restores nodes to a healthy status. However, certain hardware or software issues can cause the node to remain unhealthy for extended periods of time.

Administrators can monitor node status using the Nodes tab in the Diamanti UI. Administrators can also use the following commands to display the node health status:

```
$ dctl node health status <node-name>
$ dctl node network status <node-name>
```

Monitoring Events

The Diamanti software displays a list of events that you can use to monitor and manage resources in the cluster. You can also configure policies to specify the number of days to retain events and enable SNMP traps, as required.

Use the following command to display events:

```
$ dctl event list
```

By default, the 50 most recent events are displayed. You can use command options to list up to 1000 events, and optionally specify an offset allowing you to page through a larger number of events. You can also filter events by node, pod, or volume.

Use the following command to configure event policies:

```
dctl event configure --retention-days <n>
--snmp-community "<string>" --snmp-enable=<true|false>
--snmp-receiver=<ip-address-1[,ip-address-2,ip-address-3]>
```

For example:

```
$ dctl event configure --retention-days 60 --snmp-enable=true
--snmp-receiver=192.168.0.111
```

This command configures an event policy enabling SNMP traps with 60 day retention for events.

The following shows how to configure multiple SNMP receivers:

```
$ dctl event configure --retention-days 60 --snmp-enable=true
--snmp-receiver=192.168.0.111,192.168.0.112,192.168.0.113
```

Finally, you can display the current event configuration using the following command:

```
dctl event status
```

Deploying Workloads

After creating Diamanti network and storage objects, you can deploy workloads as containers using Kubernetes pod definitions. For more information about deploying workloads, refer to *Diamanti D-Series Technical Note: Exploring Appliance Capabilities*.

Index

Legal Notices

Publication Date: This document was published on January 7, 2020.

Publication Number: DM-QSG-20200107-01

Copyright

Trademarks