Liqid Kubernetes Integration Utility

v3.4

Copyright (c) 2024 by Liqid, Inc. - All Rights Reserved

Table of Contents

[Liqid and Kubernetes 3](#__RefHeading___Toc142_4286456116)

[Principles of Operation 5](#__RefHeading___Toc144_4286456116)

[Accessing Kubernetes 5](#__RefHeading___Toc685_1845371757)

[Accessing the Liqid Cluster 5](#__RefHeading___Toc687_1845371757)

[Tying Liqid Compute Nodes to Kubernetes Worker Nodes 6](#__RefHeading___Toc689_1845371757)

[Desired Configuration State 7](#__RefHeading___Toc691_1845371757)

[Generic Resource Specification 7](#__RefHeading___Toc693_1845371757)

[Vendor Resource Specification 7](#__RefHeading___Toc695_1845371757)

[Vendor/Model Specification 8](#__RefHeading___Toc697_1845371757)

[Support for Multi-Tenancy 11](#__RefHeading___Toc169_4286456116)

[Typical Workflow 12](#__RefHeading___Toc169_4286456116_Copy_1)

[Usage 13](#__RefHeading___Toc98_4286456116)

[Invoking the Utility 13](#__RefHeading___Toc123_4286456116)

[Logging (-l switch) 13](#__RefHeading___Toc125_4286456116)

[Ignoring Warnings (-f switch) 14](#__RefHeading___Toc127_4286456116)

[Dry-run execution (-no switch) 14](#__RefHeading___Toc129_4286456116)

[Common Switches 14](#__RefHeading___Toc134_4286456116)

[-ip={ip\_address} 14](#__RefHeading___Toc136_4286456116)

[-u={username} 14](#__RefHeading___Toc148_4286456116)

[-p={password} 15](#__RefHeading___Toc150_4286456116)

[-px={url} 15](#__RefHeading___Toc138_4286456116)

[Command Reference 16](#__RefHeading___Toc100_4286456116)

[Adopt 16](#__RefHeading___Toc102_4286456116)

[Annotate 17](#__RefHeading___Toc104_4286456116)

[Machine annotations 17](#__RefHeading___Toc159_4286456116)

[Resource annotations 17](#__RefHeading___Toc161_4286456116)

[Automatic Annotation 19](#__RefHeading___Toc163_4286456116)

[Clearing Annotations 20](#__RefHeading___Toc165_4286456116)

[Compose 21](#__RefHeading___Toc106_4286456116)

[Initialize 22](#__RefHeading___Toc108_4286456116)

[Link 24](#__RefHeading___Toc110_4286456116)

[Nodes 25](#__RefHeading___Toc112_4286456116)

[Release 26](#__RefHeading___Toc114_4286456116)

[Reset 27](#__RefHeading___Toc116_4286456116)

[Resources 28](#__RefHeading___Toc118_4286456116)

[Unlink 29](#__RefHeading___Toc120_4286456116)

# Liqid and Kubernetes

A full description of Kubernetes is far beyond the scope of this document; however, a short comment may be useful. Kubernetes provides orchestration for organizations which require the deployment and management of many -- perhaps hundreds of -- containers. Kubernetes will schedule the containers (as pods) on various servers in the Kubernetes inventory based on the available capacity of computing cores, memory, or other resources attached to those servers.

It is this interaction based on resources in which we are interested.

A Kubernetes deployment is certainly able to find creative ways to allocate containers based on server resources. However, the profiles of resources per server are usually static, which imposes some limitations on the ways in which Kubernetes may position the various containers.

Liqid Composability provides the ability to dynamically allocate PCI-attached devices such as SSDs or GPUs, among various Liqid-managed servers. What this means, is that administrators can evaluate their inventory of Kubernetes servers and reallocate resources, if necessary to assist Kubernetes in rebalancing the containers which are being managed.

For example, a general configuration might allocate 5 GPUs per server, allowing Kubernetes to allocate up to five services or jobs requiring one GPU each, upon each of those servers. However, it may eventuate that a single service or job may need to be scheduled which required (for some reason) ten GPUs. In a typical Kubernetes deployment, that service or job would not be schedulable by Kubernetes.

However, Liqid Composability would allow the administrator to dynamically compose a machine containing a Kubernetes worker node with ten GPUs, allocating one from each of five other servers. Once the GPUs are online and detectable, Kubernetes can then schedule that service or job.

A challenge arises in doing so. It is inadvisable to simply pull a GPU from a running Kubernetes worker with services or jobs currently running on that worker node. While the GPU can be hot-plugged away from the server, any service or job which might currently be using that GPU would likely crash. While Kubernetes could recover from this situation, it is better to cordon the worker node, making it unschedulable, and then cleanly evicting the various pods on that node, before removing the GPUs.

In a process which might affect multiple servers, this can become messy, even with (and sometimes due to) the automatic responses which will be provided by Kubernetes in the effort to keep the various deployments online.

The Kubernetes Integration utility automates this process. In alignment with Kubernetes' preference for analyzing applications' requested states in an effort to attain and preserve those states, the Liqid Integration Utility also observes requested state, and will, when requested, automatically perform all actions necessary, including the management of Kubernetes worker nodes, to effect that state if the current configuration deviates from that state.

# Principles of Operation

The integration utility is designed around the concept of having a desired configuration state stored in a programmatically accessible location, which can be compared against the current state of a Kubernetes deployment and a corresponding Liqid Cluster.

This implies that the following are available to the utility:

* Management access to the Kubernetes deployment
* Configuration access to the Liqid Cluster Directory
* The desired configuration state, at least as concerning resource allocations to worker nodes

As well as what is listed in the previous table, there must be some means of identifying the Liqid Cluster compute nodes' associations with the Kubernetes worker nodes. The former are identified by resource names, the latter by user-defined identifiers. Note that there is no requirement for knowledge of the actual OS host names.

## Accessing Kubernetes

Management access to the Kubernetes deployment is facilitated by providing the URL for a Kubernetes kubectl proxy. Documentation describing the usage of kubectl as a proxy is available elsewhere, and should be carefully consulted before implementing a proxy. For most usages of the utility, the proxy URL is required, and is specified on the command line.

The kubectl proxy accepts requests from the utility, and redirects them to one of potential several Kubernetes control plane servers after properly securing the communications. The traffic between the utility and the proxy are *not* secured, either by credentials or by encryption.

For this reason, Liqid recommends that the proxy be implemented on the same host which runs the utility, and only for so long as the utility needs to be invoked. In such a configuration, the kubectl proxy can be limited to accept requests only from the localhost, thus mitigating the security concerns.

## Accessing the Liqid Cluster

Access to the Liqid Cluster is done via the REST API front-end of the Liqid Director. Specifics regarding the director are available elsewhere, but a few points are noteworthy.

The utility requires the IP address of the Liqid Director for certain commands. Generally, the IP address (among other datums) are stored in the Kubernetes database, and are retrieved by the utility as necessary.

The Liqid Director recognizes several modes of access security. The utility will not work if the Liqid Director is secured via LDAP or OpenID. Only basic (or no) authentication is allowed.

The credentials used to communicate with the Liqid Director can be kept in the Kubernetes database as a secret. While these credentials are not stored in clear-text, they are easily retrieved, and may be extracted by a malicious user. Liqid suggests that these credentials should *not* be stored, and should instead be provided by the administrator on each invocation of the utility.

## Tying Liqid Compute Nodes to Kubernetes Worker Nodes

As mentioned, Kubernetes persists a user-defined identifier for each of the worker nodes which is a convenient means of specifying the nodes via kubectl. This node name is maintained in the metadata portion of the node struct, and may be displayed using either of the following commands:

* kubctl describe nodes
* k8s-integration nodes

The Liqid Cluster has no knowledge of the Kubernetes worker names; indeed, it has no knowledge either, of the host name defined within the operating system loaded on the worker node. What the Cluster does have, is an arbitrary name for the compute resource which represents the server hosting the worker node. That name is formatted as "pcpu{n}" where {n} is a monotonically-increasing integer value, starting at zero.

There is no programmatic method of deriving the node name from the Liqid resource name, nor vice-versa. To enable the utility to function, the following information must be specified, per node:

The name of the Liqid Machine which acts as a logical container (from the Liqid perspective) of the compute node as well as the attached resources, must be stored as an annotation on the corresponding worker node:

kubint-liqid.com/machine-name=pcpu0-worker-4

This assumes the Kubernetes worker node name is worker-4, and that the machine name is pcpu0-worker-4 (the name of the machine implies that the compute node is named pcpu0).

The name of the worker node must additionally be stored as the user description for the Liqid compute device representing the worker node.

This information must be provided manually, however this can be effected via the utility's Annotate command, which applies both the Kubernetes annotation and the Liqid user description.

## Desired Configuration State

The desired configuration state is stored as annotations on each of the worker nodes in the Kubernetes deployment. The desired state for any particular worker node is described by a set of resource specifications for each general type of PCI-connected resource as described below:

|  |  |  |
| --- | --- | --- |
| Annotation Key | Resource Type | Annotation Value |
| kubint.liqid.com/fpga-resources | FPGAs | {specification\_list} |
| kubint.liqid.com/gpu-resources | GPUs | {specification\_list} |
| kubint.liqid.com/link-resources | LINKs | {specification\_list} |
| kubint.liqid.com/memory-resources | Memory Devices | {specification\_list} |
| kubint.liqid.com/ssd-resources | SSDs | {specification\_list} |

The format of the annotation value is the same for all annotation types -- a comma-separated list of specifications of resources and counters.

### Generic Resource Specification

The Generic Specification is the simplest, being comprised only of a single integer. It declares a particular number of resources of a given type which should be attached to the corresponding worker node. For example, the following annotation:

kubint.liqid.com/gpu-resources=5

would indicate that the corresponding worker node should have 5 GPUs attached thereto. In many cases, no further additional specifications should be necessary.

If there is no annotation for a particular resource type, the implication is that the worker node should have no resources of that type attached to it.

### Vendor Resource Specification

The Vendor Specification is more specific. It is formatted as a vendor name string and a counter, inter-delimited by a colon character. For example:

kubint.liqid.com/gpu-resources=ACME Corporation:3

Indicates that 3 GPUs made by the ACME corporation should be attached to the worker node. The presumption is that the Liqid Cluster contains GPUs from multiple vendors, and that this particular worker node requires GPUs only from the ACME corporation. This specification restricts the allocation to only those from ACME.

Note the embedded space in the vendor name string. While ACME Corporation is a fictional company name, many, if not most, vendor names do contain embedded spaces. This should be kept in mind when typing the utility command line - the administrator should ensure delimiting quotation characters are used if and where necessary.

Supposing a worker node requires 5 GPUs, but only 2 of them must be from ACME, while the other 3 can be from any vendor. The following annotation will specify exactly that requirement:

kubint.liqid.com/gpu-resources=ACME Corporation:2,3

The two specifications (separated by a comma) indicate a requirement of 2 ACME GPUs, and 3 more GPUs from any vendor.

Supposing that a worker node requires 10 GPUS - 2 from ACME, 3 from Amalgamated, and 5 more from any vendor. The following specification would be used:

kubint.liqid.com/gpu-resources=ACME Corporation:2,Amalgamated Inc:3,5

Finally, one can specify a counter of zero in a vendor specification, to indicate that regardless of any other specifications, no resources of the indicated type from the indicated vendor should be allocated. The following specification:

kubint.liqid.com/gpu-resources=ACME Corporation:0,10

Indicates that 10 GPUs are to be attached to the worker node, from any available vendor *except* ACME.

### Vendor/Model Specification

The Vendor/Model specification provides a counter which relates to a specific vendor AND model. It contains three parts, inter-delimited by colon characters:

{vendor\_name} : {model\_name} : {count}

To indicate that a worker node requires 4 ACME model G200 GPUs, the following specification would be used:

kubint.liqid.com/gpu-resources=ACME Corporation:G200:4

If a worker node requires 3 model G200 and 3 model G300 GPUs, one would specify:

kubint.liqid.com/gpu-resources=ACME Corporation:G200:3,ACME Corporation:G300:3

Finally, consider the following:

kubint.liqid.com/gpu-resources=ACME:0,Initrode:ITRD30:3,Initech:ITCH20:0,10

This specification indicates that the worker node requires 3 Initrode model ITRD30 GPUs, and 10 more GPUs of any type and model excluding Initech:ITCH20 and excluding all models from ACME.

Note that the order of specifications in a specification list is irrelevant. The meaning of a properly-constructed specification list does not change with different orderings of the comprising specifications.

Also note that the vendor names and model strings must correspond exactly to what is reported by the Liqid Director. This information is available via the Resources command, described in a later section of this document.

With the above information in place, the utility is now able to function properly. The basic invocation of most commands can be described by the following steps:

* The administrator enters the command
* The utility parses the input, and proceeds accordingly
* The Kubernetes control plane is accessed
* Liqid Director information (possibly including credentials) are taken from the Kubernetes database
* The Liqid Cluster is accessed via the REST API for the Liqid Director
* The current state of the deployment in terms of both Kubernetes and the Liqid Cluster are ascertained
* The command is evaluated, and a plan is created consisting of one or more steps which might alter any of the following:
  + The annotations on various worker nodes
  + The general Liqid-specific information in the Kubernetes database
  + The general descriptions of compute nodes in the Liqid Cluster
  + The schedulability of various worker nodes, and the eviction of pods on those nodes
  + The composition of the various Liqid machines corresponding to those nodes
* Various errors or warnings are displayed
* The plan is displayed (if no errors are found, or if the -f switch was specified)
* The plan is executed (assuming that the -no switch was not specified)

# Support for Multi-Tenancy

The utility does respect the following considerations:

* The Liqid Cluster may be hosting resources and compute nodes which are not part of the Kubernetes Cluster
* The Kubernetes deployment may have worker nodes which are not participating in Liqid composability
* There may be multiple Kubernetes deployments using a single Liqid Cluster

To address all of these considerations, the utility expects and requires that all compute nodes and resources which are candidates for composition by the utility, to be assigned to one distinct Liqid Cluster group which is reserved specifically for one particular Kubernetes deployment.

All Liqid resources which are in the System free pool, or in other group free pools or are attached to machines which are owned by other groups, are not considered by the utility.

Further, all nodes in the Kubernetes deployment which do not have Liqid-specific annotations are not considered by the utility.

In consideration of this, part of the initial configuration requires the following actions:

* A group must be created
* The name of the group must be stored in the Liqid-specific config-set in Kubernetes (along with the IP address of the Liqid Director and optionally the credentials - as previously described)
* All relevant resources (including compute resources) must be moved to the group free pool
* A machine must be created for each of the compute resources
* The corresponding worker nodes must be annotated with the machine name (as previously described)
* The corresponding compute resource must have a user description indicating the node name of the worker node (as previously described)

By isolating the resources within the Kubernetes-specific group, the utility is able to confine its actions to those resources, avoiding collisions with other tenants of the Liqid Cluster.

# Typical Workflow

A Kubernetes deployment is created (potentially including only a control plane at this point)

A Liqid Cluster is deployed per convention

Liqid Cluster compute nodes are configured with an appropriate operating system

The compute nodes are configured with all 3rd-party software, and are subsequently adopted into the Kubernetes deployment as worker nodes

The Initialize command is invoked, which creates a Liqid group, moves all resources into that group, stores Liqid Cluster information in the Kubernetes database, creates user descriptions on the resource nodes, and annotates the Kubernetes worker nodes.

The Initialize command may also create the initial set of resource annotations, and compose the various resources accordingly.

As requirements emerge or change, resource annotations are updated and the Compose command is invoked, which applies the changed annotations via composing resources accordingly

# Usage

## Invoking the Utility

The integration utility is implemented as a Java program and is provided as a standard .jar file, which may be executed by any recent JRE. Liqid recommends openjdk version 20.

The utility may be invoked by typing the following:

java -jar k8s-integration.jar {switches} {command}

A select number of commands have been implemented, and each command accepts and in some cases requires a particular set of switches. The entire selection of switches and commands can be obtained by typing:

java -jar k8s-integration.jar --help

The particular version of the utility may be obtained by typing:

java -jar k8s-integration.jar --version

Command Line Switches

Command line switches generally follow the conventions established for unix utilities. They may be specified in their long or short forms, and those switches which accept argument values may be specified either as:

-switch=value

or as:

-switch value

A few switches will accept multiple values, and can be specified as:

-switch=value1,value2,...

or as:

-switch value1 -switch value2 ...

## Logging (-l switch)

Certain diagnostic information may be written to stdout and/or to stderr. More specific diagnostics can be enabled by adding the -l switch, which enables verbose logging. Such logging will be placed in a file named "liq.config.log."

During trouble-shooting, Liqid support personnel may request that the -l switch is enabled, and that the resuling log file is sent to Liqid for analysis.

## Ignoring Warnings (-f switch)

While preparing to carry out user requests, the utility may detect certain speficiation or configuration anomalies which could produce unexpected results. When this happens, an error message is displayed and the utility stops before taking any actions. It may be the case that several messages are displayed, but *generally*, no action will be taken when such anomalies are detected.

This behavior can be overridden with the -f switch, which forces action even in the presence of such anomalies. This switch is to be used with caution.

## Dry-run execution (-no switch)

Most commands accept the -no option, which prevents the utility from taking any action. The utility \*will\* analyze command line input as well as Liqid and Kubernetes Cluster configuration states, it \*will\* create a plan for altering the Cluster states, but it will stop before actually executing the plan.

## Common Switches

Many comands share a certain subset of switches. These are listed below:

### -ip={ip\_address}

The -ip switch accepts an argument representing the IP address or DNS name for the Liqid Cluster Director. This is configured at install time for the Liqid Cluster, and is necessary for this utility to communicate with the Liqid Cluster to check the configuration of the Cluster, and to effect any necessary configuration changes.

### -u={username}

The -u switch specifies a username credential which will be used by this utility when communicating with the Liqid Cluster. It is expected that the Cluster is configured to use either no authentication or local username/password authentication. LDAP and OpenID are not currently supported.

If the Liqid and Kubernetes Clusters are linked with credentials, then the credentials stored with Kubernetes are normally used for accessing the Liqid Cluster. Specifying this switch will override such behavior.

### -p={password}

The -p switch specifies a password credential which will be used by this utility when communicating with the Liqid Cluster. It is expected that the Cluster is configured to use either no authentication or local username/password authentication. LDAP and OpenID are not currently supported.

If the Liqid and Kubernetes Clusters are linked with credentials, then the credentials stored with Kubernetes are normally used for accessing the Liqid Cluster. Specifying this switch will override such behavior.

### -px={url}

The -px switch accepts an argument representing the URL for a kubectl proxy. Information regarding the specification and use of kubectl proxy is available elsewhere, but generally it exists to allow simplified access to the Kubernetes API. This utility leverages the kubectl proxy facility for communication with Kubernetes.

Ensure that the proxy is configured such that it will accept all API requests from the host on which this utility is running.

Be aware that the use of kubectl proxy carries certain security concerns. Ensure that you are taking proper steps to secure your Kubernetes Cluster when using kubectl proxy.

Example invocation for the kubectl proxy:

kubectl proxy --port=8080

If the kubectl proxy is running on the same host as is the utility, then the -px switch might look like:

-px=http://127.0.0.1:8080

# Command Reference

This section describes the various implemented commands, providing information on how, why, and when to use each command.

## Adopt

This command is used when new resources are added to the Liqid Cluster, and are to be made available to this utility. It is presumed that a Liqid and Kubernetes cluster has already been established and is being managed via annotations and by this utility.

For compute nodes, the following actions are taken:

* A Liqid Machine is created for the compute node, and is given a name which reflects both the resource name (formatted as "pcpu{n}") and the Kubernetes worker name
* The Liqid compute node resource will be linked with Kubernetes via a user description in the Liqid resource, which contains the name of the Kubernetes worker node
* The Kubernetes worker will be linked with the Liqid Machine via an annotation on the Kubernetes worker node which contains the Liqid Machine name

The compute node is expected to already be participating as a worker node in the Kubernetes cluster, with all relevant software and plug-ins installed, including (but not limited to) kubelet and kube-proxy.

For other resource nodes such as GPUs or FPGAs, the following action is taken:

* The resource is moved into the group free pool for the Liqid Group which is associated with the Kubernetes Cluster.

The resource is now eligible for automatic composition via the compose command.

Syntax:

adopt

-px,--proxy-url={proxy\_url}

[ -pr,--processors={pcpu\_name:worker\_node\_name}[,...] ]

[ -r,--resources={name}[,...] ]

[ -f,--force ]

[ -no,--no-update ]

For example, if you have added a compute node identified as "pcpu5" and it is known to Kubernetes as "worker-5", you would type something like:

java -jar k8s-integration.jar -px=http://127.0.0.1:8080 -pr=pcpu5:worker-5

If you added several GPU nodes identified as "gpu7", "gpu8", and "gpu9", you might type:

java -jar k8s-integration.jar -px=http://127.0.0.1:8080 -r=gpu7,gpu8,gpu9

## Annotate

This command is used to indicate, for a particular Kubernetes worker node, the Liqid Resource specifications which apply to that node. It is provided as a convenient alternative to requiring the adminsitrator to apply the annotations manually via kubectl.

The Compose command relies upon the various worker node annotations for working out which devices should be composed into which machines.

### Machine annotations

There is one machine annotation per worker node. It is formatted as:

kubint.liqid.com/machine-name={machine\_name}

This annotation ties the worker node to the Liqid Machine which contains the compute node associated with the worker node itself. The Liqid Machine will contain one compute resource, and potentially many other resources such as GPUs and FPGAs.

### Resource annotations

A resource specification consists of a list of resource model specifications which describe, in aggregate, the general profile of the resources which should be composed into the machine containing the given worker node.

A resource specification is made up of one, two, or three parts, and applies to a specific resource type (FPGA, GPU, MEMORY, LINK, or SSD).

A Generic Resource Specification consists of only a count, representing the number of such resources from any vendor, and of any model, which should be composed into the containing machine. It is an integer formatted simply as:

{count}

A Vendor Resource Specification consists of a vendor name and a count, indicating the number of such resources from that vendor, which should be composed into the containing machine. It is formatted as:

{vendor\_name}:{count}

A Specific Resource Specification consists of a vendor name, model, and count, indicating the number of such resources from that vendor and of that model, which should be composed into the containing machine. It is formatted as:

{vendor\_name}:{model\_name}:{count}

Resource specifications are generally additive. That is, if one specifies

ACME:T4:2,ACME:3,2

meaning a count of 2 ACME T4 GPUs, 3 more ACME GPUs of any model, and finally 2 more GPUs of any vendor and model, the total specification is for 2 + 3 + 2 = 7 GPUs. They might all be ACME T4s, but there will certainly be at least two of T4s as well as three more (for a total of five) ACME GPUs, and potentially as many as 7, although the last two most general GPUs could also be from INITECH or INITRODE.

One may also specify alternative vendors and models, for example:

INITECH:IH-4:3,INITRODE:ID-1000:2,3

indicating 3 INITECH IH-4 FPGAs, 2 INITRODE ID-1000 FPGAs, and 3 FPGAs of any vendor.

There is a special case where a count of zero can be used in conjunction with other specifications.

A Specific Resource Specification of zero for a particular vendor and model indicates that no resources of the indicated type matching that vendor and model should be composed into the machine. This could be presented in order to allow all models from a vendor except for a particular model. In other words, one could specify:

ACME:5,ACME:T500:0

which means 5 SSDs from ACME, but none of the ACME T500 models. In this case, 5 SSDs would be composed into the machine of any model from ACME except for T500s.

A Vendor Resource Specification of zero indicates that no resources of the indicated type from that vendor should be composed into the machine. For example:

12,INITECH:0

indicating that twelve GPUs of any vendor and model should be composed into the machine, excepting that no GPUs from INITECH should be allowed.

Finally, a Generic Resource Specification of zero indicates that no resources of the indicated type should be composed into the machine at all. This special case does not exist in the actual worker node annotations; rather, it is specified on the command line to indicate that any existing annotation of that type should be removed. This would prevent any resources of that type from being composed into the machine.

These types are given as annotations on the relevant worker node formatted as:

kubint.liqid.com/fpga-resources={fpga\_spec},...

kubint.liqid.com/gpu-resources={gpu\_spec},...

kubint.liqid.com/link-resources={link\_spec},...

kubint.liqid.com/memory-resources={mem\_spec},...

kubint.liqid.com/ssd-resources={ssd\_spec},...

A complete specification be annotated as:

kubint.liqid.com/gpu-resources=ACME:T200:5,ACME:2

kubint.liqid.com/link-resources=NETZ:T25:2,NETZ:10,5

and it could be effected by the following:

java -jar k8s-integration.jar -px=http:127.0.0.1:8080 -n=worker5 -m=pcpu5-worker5 \

-gs=ACME:T200:5,ACME:2 \

-ls=NETZ:T25:2,NETZ:10,5 \

-fs=0

Syntax:

annotate

-px,--proxy-url={proxy\_url}

-n,--worker-node={worker\_node\_name}

[ -m,--liqid-machine={liqid\_machine} ]

[ -fs,--fpga-spec={spec}[,...] ]

[ -gs,--gpu-spec={spec}[,...] ]

[ -ls,--link-spec={spec}[,...] ]

[ -ms,--mem-spec={spec}[,...] ]

[ -ss,--ssd-spec={spec}[,...] ]

[ -f,--force ]

[ -no,--no-update ]

### Automatic Annotation

A special case of this command exists which evaluates the inventory of resources assigned to the relevant Liqid Cluster Group or to machines within that group, allocating them as evenly as possible across all of the properly-labeled worker nodes.

For example, if there are a total of 15 GPUs and 8 FPGAs, and there are 5 worker nodes in the configuration, the automatic algorithm would assign 3 GPUs to each of the worker nodes and either 1 or 2 FPGAs to each of the worker nodes.

No resources will actually be moved; no composition takes place. Rather, annotations are simply rewritten to reflect the calculated allocations.

Syntax:

annotate

-px,--proxy-url={proxy\_url}

-a,--automatic

[ -f,--force ]

[ -no,--no-update ]

### Clearing Annotations

A special case of this command exists which is designed to clear all annotations from a particular worker node. This includes annotations for all resource specifications. It does NOT clear the machine annotation.

Syntax:

annotate

-px,--proxy-url={proxy\_url}

-n,--worker-node={worker\_node\_name}

-cl,--clear

[ -f,--force ]

[ -no,--no-update ]

## Compose

This command evaluates the difference between the current composition of the Liqid Machines and the desired composition as defined by the Kubernetes worker node annotations, develops a plan for composing the various machines in as efficient and least-disruptive manner as possible, then executes that plan.

The algorithm for determining which reconfigurations are necessary is complex, but it is designed to allow a typical Kubernetes configuration to continue operating with little or no production downtime.

Specifically, adding resources to a machine has little effect on the corresponding worker node. However, removing resources requires that the worker node becomes unschedulable, and all pods on the machine must be evicted. Once the resource(s) is/are removed, the worker node is made schedulable, and Kubernetes will eventually bring up pods on that machine again.

This is done due to the potential deleterious effects on any particular pod resulting from suddently having a required resource taken away from it.

It is suggested that this command be executed first with the -no switch, so that the adminstrator can examine the proposed plan before allowing the utility to go ahead with the compose function.

Syntax:

compose

-px,--proxy-url={proxy\_url}

[ -f,--force ]

[ -no,--no-update ]

## Initialize

This command creates the linkage and annotations necessary for eventual resource composition management. The following conditions should exist prior to invoking the command:

* The Kubernetes deployment is complete and functioning properly
* The Liqid Cluster compute nodes which are intended to participate in the Kubernetes deployment as worker nodes, are in fact properly deployed as workers.
* The Liqid Cluster compute nodes are currently not composed into machines, nor are they assigned to any Liqid Cluster group free pool.
* The Liqid Cluster resources (such as FPGAs and GPUs) which are to participate in the Kubernetes deployment are not currently composed into any machines, nor are they assigned to any Liqid Cluster group free pool.
* The Liqid Cluster group which is intended to hold the various resources and compute nodes has not yet been created.

Certain of these requirements can be waived if you use the -f switch. For example, the resources will be removed from machines and/or groups, the identifier group will be removed, and other certain actions will be taken to ensure a clean configuration prior to initialization. None of these additional actions will be taken in the absence of the -f switch.

When this command is invoked, the following major actions are undertaken:

* Any conflicting configuration state will be resolved, but only if the -f switch is specified
* A linkage will be created which stores Liqid Cluster information and (optionally) Liqid credentials in the Kubernetes database.
* A group will be created to contain the various machines which will be composed for the Kubernetes deployment
* Liqid machines will be composed within the Liqid Cluster group, per compute node
* Liqid Cluster compute nodes will have descriptions applied to them, linking the node to the corresponding Kubernetes worker name
* Kubernetes nodes will be annotated to refer to the Liqid Cluster machines containing the corresponding compute nodes
* The called-out resources (such as FPGAs and GPUs) will be assigned to the Liqid Cluster group free pool

Additionally, the -al switch may be specified which causes the following additional actions to be taken:

The worker nodes will be annotated with resource specifications which allocate resources across the worker nodes, as evenly as possible

Those resources will be composed into the corresponding Liqid Cluster machines, such that they become available for use by the various pods in the corresponding worker nodes

Any clients of the Liqid Cluster apart from the Kubernetes Cluster in question, will not be affected. Specifically, any resources or compute nodes which are not specified for this command, will be ignored.

Specifying compute nodes

The compute nodes are specified as a list of arguments to the -pr switch. Each specification is presented as a colon-seperated pair of identifiers:

compute\_node\_name : worker\_node\_name

For example, the following specification:

-pr=pcpu0:worker-0,pcpu1:worker-1,pcpu2:worker-2

would be appropriate for indicating that the Liqid compute node identified as pcpu0 is to be associated with the Kubernetes woker node named worker-0, pcpu1 with worker-1, and pcpu2 with worker-2. Again, it is expected that the compute nodes are already installed and deployed as Kubernetes worker nodes with the indicated Kubernetes node names.

The composable resources are specified as a simple list of resource names. For example, four GPUs and two FPGAs would likely be specified as:

-r=gpu0,gpu1,gpu2,gpu3,fpga0,fpga1

Syntax:

initialize

-px,--proxy-url={proxy\_url}

-ip,--liqid-ip-address={ip\_address}

[ -u,--liqid-username={user\_name} ]

[ -p,--liqid-password={password} ]

-g,--liqid-group={group\_name}

-pr,--processors={pcpu\_name:worker\_node\_name}[,...]

-r,--resources={name}[,...]

[ -al,--allocate ]

[ -f,--force ]

[ -no,--no-update ]

## Link

Before using the Compose command, it is necessary to link the Kubernetes deployment with the Liqid Cluster. This consists mainly of recording a small amount of information regarding how to access the Liqid Cluster Director in the Kubernetes database.

While this could be done manually, the utility provides the Link command for carrying out this process more easily.

When invoked, the command will:

* Record the IP address of the Liqid Cluster Director in the Kubernetes database
* Record the name of the Liqid Cluster group associated with this Kubernetes deployment in the Kubernetes database
* Optionally record credentials for accessing the Liqid Cluster Director as a secret, in the Kubernetes database

Syntax:

link

-px,--proxy-url={proxy\_url}

-ip,--liqid-ip-address={ip\_address}

[ -u,--liqid-username={user\_name} ]

[ -p,--liqid-password={password} ]

-g,--liqid-group={group\_name}

[ -f,--force ]

[ -no,--no-update ]

## Nodes

This command lists all of the worker nodes from the perspective of Kubernetes. Of particular interest would be the pods currently assigned to the node, as well as the Liqid-specific annotations.

The -f and -no switches are not supported as this command makes no configuration changes to either the Liqid Cluster nor to Kubernetes.

Syntax:

nodes

-px,--proxy-url={proxy\_url}

## Release

This command effectively removes knowledge of Liqid Cluster resources from Kubernetes. Use this when compute nodes or other resources are taken out of the Liqid Cluster, or are reassigned for other uses.

Supposing one wishes to stop using two GPUs and a compute node, one might type:

java -jar k8s-integration.jar release -px=http://127.0.0.1:8080 -r=gpu4,gpu5,pcpu3

Syntax:

release

-px,--proxy-url={proxy\_url}

-r,--resources={name}[,...]

[ -f,--force ]

[ -no,--no-update ]

## Reset

Do not use this command.

It will remove all Liqid annotations from all worker nodes.

It will remove Liqid linkage information (Liqid IP address and group name) from the Kubernetes database.

It will completely unconfigure the Liqid Cluster, including any groups and machines which have nothing to do with the Kubernetes deployment.

If you do use this command, use the -no switch first to see the reletively short (but very dangerous) plan which will be created. Then don't use it anyway.

Syntax:

reset

-px,--proxy-url={proxy\_url}

-ip,--liqid-ip-address={ip\_address}

[ -u,--liqid-username={user\_name} ]

[ -p,--liqid-password={password} ]

-f,--force

[ -no,--no-update ]

## Resources

This command lists the Liqid Cluster inventory. Relevant information includes the names, vendors, and models of the various resources, and the current assignation of such resources to the various machines.

The -f and -no switches are not supported as this command makes no configuration changes to either the Liqid Cluster nor to Kubernetes.

Syntax:

resources

-ip,--liqid-ip-address={ip\_address}

[ -u,--liqid-username={user\_name} ]

[ -p,--liqid-password={password} ]

## Unlink

This command reverses the action of the Link command.

Specifically, it removes the Kubernetes ConfigMap entries which contain the Liqid Cluster information for the cluster associated with this Kubernetes deployment, as well as the secret Liqid credentials if they are stored in the Kubernetes database.

This command does not remove annotations from the worker nodes.

Use this command when you no longer intend to use this utility for automatic composition of Liqid resources in conjunction with the Kubernetes deployment, or when you wish to re-establish the information from scratch (although the Link command would do this anyway).

Syntax:

unlink

-px,--proxy-url={proxy\_url}

[ -f,--force ]

[ -no,--no-update ]