Inspect Kubernetes Resources

Objectives

- Query, format, and filter attributes of Kubernetes resources.

Kubernetes and OpenShift Resources

Kubernetes uses API resource objects to represent the intended state of everything in the cluster. All administrative tasks require creating, viewing, and changing the API resources. Use the `oc api-resources` command to view the Kubernetes resources.

> **NOTE**
>
> The `oc` commands in the examples are identical to the equivalent `kubectl` commands.

```
[user@host ~]$ oc api-resources
NAME                    SHORTNAMES   APIVERSION   NAMESPACED   KIND
bindings                             v1           true         Binding
componentstatuses       cs           v1           false        ComponentStatus
configmaps              cm           v1           true         ConfigMap
endpoints               ep           v1           true         Endpoints
...output omitted...
daemonsets              ds           apps/v1      true         DaemonSet
deployments             deploy       apps/v1      true         Deployment
replicasets             rs           apps/v1      true         ReplicaSet
statefulsets            sts          apps/v1      true         StatefulSet
...output omitted...
```

The `SHORTNAME` for a component helps to minimize typing long CLI commands. For example, you can use `oc get cm` instead of `oc get configmaps`.

The `APIVERSION` column divides the objects into API groups. The column uses the `<API-Group>/<API-Version>` format. The `API-Group` object is blank for Kubernetes core resource objects.

Many Kubernetes resources exist within the context of a Kubernetes namespace. Kubernetes namespaces and OpenShift projects are broadly similar. A one-to-one relationship always exists between a namespace and an OpenShift project.

The `KIND` is the formal Kubernetes resource schema type.

The `oc api-resources` command can further filter the output with options that operate on the data.

Table 2.1. The `api-resources` Command Options

Option Example	Description
`--namespaced=true`	If false, return non-namespaced resources, otherwise return namespaced resources
`--api-group apps`	Limit to resources in the specified API group. Use `--api-group=''` to show core resources.
`--sort-by name`	If non-empty, sort the list of resources by using the specified field. The field can be either 'name' or 'kind'.

For example, use the following `oc api-resources` command to see all the `namespaced` resources in the `apps` API group, sorted by `name`.

```
[user@host ~]$ oc api-resources --namespaced=true --api-group apps --sort-by name
NAME                 SHORTNAMES   APIVERSION   NAMESPACED   KIND
controllerrevisions               apps/v1      true         ControllerRevision
daemonsets           ds           apps/v1      true         DaemonSet
deployments          deploy       apps/v1      true         Deployment
replicasets          rs           apps/v1      true         ReplicaSet
statefulsets         sts          apps/v1      true         StatefulSet
```

Each resource contains fields that identify the resource or that describe the intended configuration of the resource. Use the `oc explain` command to get information about valid fields for an object. For example, execute the `oc explain pod` command to get information about possible pod object fields.

```
[user@host ~]$ oc explain pod
KIND:     Pod
VERSION:  v1

DESCRIPTION:
     Pod is a collection of containers that can run on a host. This resource is
     created by clients and scheduled onto hosts.

FIELDS:
   apiVersion   <string>
     APIVersion defines the versioned schema of this representation of an
     ...

   kind <string>
     Kind is a string value representing the REST resource this object
     ...

   metadata     <ObjectMeta>
     Standard object's metadata. More info:
     ...

   spec <PodSpec>
     Specification of the desired behavior of the pod. More info:
     ...output omitted...
```

Every Kubernetes resource contains the `kind`, `apiVersion`, `spec`, and `status` fields. However, when you create an object definition, you do not need to provide the `status` field. Instead, Kubernetes generates the `status` field, and it lists information such as runtime status and readiness. The `status` field is useful for troubleshooting an error or for verifying the current state of a resource.

You can use the YAML path to a field and dot-notation to get information about a particular field. For example, the following `oc explain` command shows details for the `pod.spec` fields.

```
[user@host ~]$ oc explain pod.spec
KIND:     Pod
VERSION:  v1

FIELD: spec <PodSpec>

DESCRIPTION:
     Specification of the desired behavior of the pod. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

     PodSpec is a description of a pod.

FIELDS:
   activeDeadlineSeconds        <integer>
     Optional duration in seconds the pod may be active on the node relative to
    ...output omitted...
```

The following Kubernetes main resource types can be created and configured by using a YAML or a JSON manifest file, or by using OpenShift management tools:

Pods (`pod`)
Represent a collection of containers that share resources, such as IP addresses and persistent storage volumes. It is the primary unit of work for Kubernetes.

Services (`svc`)
Define a single IP/port combination that provides access to a pool of pods. By default, services connect clients to pods in a round-robin fashion.

ReplicaSet (`rs`)
Ensure that a specified number of pod replicas are running at any given time.

Persistent Volumes (`pv`)
Define storage areas for Kubernetes pods to use.

Persistent Volume Claims (`pvc`)
Represent a request for storage by a pod. PVCs link a PV to a pod so that its containers can use the provisioned storage, usually by mounting the storage into the container's file system.

ConfigMaps (`cm`) and Secrets
Contain a set of keys and values that other resources can use. Configuration maps and secrets centralize configuration values for several resources. Secrets differ from configuration maps in that the values of Secrets are always encoded (not encrypted), and their access is restricted to authorized users.

Deployment (`deploy`)
A deployment represents a set of containers that are included in a pod, and the deployment strategies to use. A `deployment` object contains the configuration to apply to all containers of each pod replica, such as the base image, tags, storage definitions, and the commands to execute when the containers start. Although Kubernetes replicas can be created stand-alone in OpenShift, they are usually created by higher-level resources such as deployment controllers.
Red Hat OpenShift Container Platform (RHOCP) adds the following main resource types to Kubernetes:

Build (`build`)
A *build* is the generic process of taking an input source, such as the source code of an application, and producing a usable resource as the output, such as a runnable image.

BuildConfig (`bc`)
A *BuildConfig* object defines a build process. A `BuildConfig` object requires at least the source input of the build, the strategy that defines how to build the input, and where to store the output of the process.

Routes
Represent a DNS hostname that the OpenShift router recognizes as an ingress point for applications and microservices.

Structure of Resources

Almost every Kubernetes object includes two nested object fields that govern the object's configuration: the object `spec` and the object `status`. The `spec` object describes the intended state of the resource, and the `status` object describes the current state. You specify the `spec` section of the resource when you create the object. Kubernetes controllers continuously update the `status` of the object throughout the existence of the object. The Kubernetes control plane continuously and actively manages every object's actual state to match the desired state you supplied.

The `status` field uses a collection of `condition` resource objects with the following fields.

Table 2.2. Condition Resource Fields

Field	Example	Description
Type	`ContainersReady`	The type of the condition
Status	`False`	The state of the condition
Reason	`RequirementsNotMet`	An optional field to provide extra information
Message	`2/3 containers are running`	An optional textual description for the condition
LastTransitionTime	`2023-03-07T18:05:28Z`	The last time that conditions were changed

For example, in Kubernetes, a `Deployment` object can represent an application that is running on your cluster. When you create a `Deployment` object, you might configure the deployment `spec` object to specify that you want three replicas of the application to be running. Kubernetes reads the deployment `spec` object and starts three instances of your chosen application, and updates the `status` field to match your `spec` object. If any of those instances fails, then Kubernetes responds to the difference between the `spec` and `status` objects by making a correction, in this case to start a replacement instance.

Other common fields provide base information in addition to the `spec` and `status` fields of a Kubernetes object.

Table 2.3. API Resource Fields

Field	Description
`apiVersion`	Identifier of the object schema version.
`kind`	Schema identifier.
`metadata.name`	Creates a label with a `name` key that other resources in Kubernetes can use to find it.
`metadata.namespace`	The namespace, or the RHOCP project where the resource is.
`metadata.labels`	Key-value pairs that can connect identifying metadata with Kubernetes objects.

Resources in Kubernetes consist of multiple objects. These objects define the intended state of a resource. When you create or modify an object, you make a persistent record of the intended state. Kubernetes reads the object and modifies the current state accordingly.

All RHOCP and Kubernetes objects can be represented as a JSON or YAML structure. Consider the following pod object in the YAML format:

```
apiVersion: v1 (1)
kind: Pod (2)
metadata: (3)
  name: wildfly (4)
  namespace: my_app (5)
  labels:
    name: wildfly (6)
spec: (7)
  containers:
    - resources:
        limits:
          cpu: 0.5
      image: quay.io/example/todojee:v1 (8)
      name: wildfly (9)
      ports:
        - containerPort: 8080 (10)
          name: wildfly
      env: (11)
        - name: MYSQL_DATABASE
          value: items
        - name: MYSQL_USER
          value: user1
        - name: MYSQL_PASSWORD
          value: mypa55
...object omitted...
status: (12)
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2023-08-19T12:59:22Z"
    status: "True"
    type: PodScheduled
```

1. Identifier of the object schema version.
2. Schema identifier. In this example, the object conforms to the pod schema.
3. Metadata for a given resource, such as annotations, labels, name, and namespace.
4. A unique name for a pod in Kubernetes that enables administrators to run commands on it.
5. The namespace, or the RHOCP project that the resource resides in.
6. Creates a label with a `name` key that other resources in Kubernetes, usually a service, can use to find it.
7. Defines the pod object configuration, or the intended state of the resource.
8. Defines the container image name.
9. Name of the container inside a pod. Container names are important for `oc` commands when a pod contains multiple containers.
10. A container-dependent attribute to identify the port that the container uses.
11. Defines a collection of environment variables.
12. Current state of the object. Kubernetes provides this field, which lists information such as runtime status, readiness, and container images.

Labels are key-value pairs that you define in the `.metadata.labels` object path, for example:

```
kind: Pod
apiVersion: v1
metadata:
  name: example-pod
  labels:
    app: example-pod
    group: developers
...object omitted...
```

The preceding example contains the `app=example-pod` and `group=developers` labels. Developers often use labels to target a set of objects by using the `-l` or the `--selector` option. For example, the following `oc get` command lists pods that contain the `group=developers` label:

```
[user@host ~]$ oc get pod --selector group=developers
NAME                          READY   STATUS    RESTARTS   AGE
example-pod-6c9f758574-7fhg   1/1     Running   5          11d
```

Command Outputs

The `kubectl` and `oc` CLI commands provide many output formatting options. By default, many commands display a small subset of the most useful fields for the given resource type in a tabular output. Many commands support a `-o wide` option that shows additional fields.

Table 2.4. Tabular Fields

`oc get pods`	`oc get pods -o wide`	**Example value**
NAME	NAME	example-pod
READY	READY	1/1
STATUS	STATUS	Running
RESTARTS	RESTARTS	5
AGE	AGE	11d
	IP	10.8.0.60
	NODE	master01
	NOMINATED NODE	<none>
	READINESS GATES	<none>

To view all the fields that are associated with a resource, the `describe` subcommand shows a detailed description of the selected resource and related resources. You can select a single object by name, or all objects of that type, or provide a name prefix, or a label selector.

For example, the following command first looks for an exact match on the `TYPE` object and the `NAME-PREFIX` object. If no such resource exists, then the command outputs details for every resource of that type with a name with a `NAME_PREFIX` prefix.

```
[user@host ~]$ oc describe TYPE NAME-PREFIX
```

The `describe` subcommand provides detailed human-readable output. However, the format of the `describe` output might change between versions, and thus is not recommended for script development. Any scripts that rely on the output of the `describe` subcommand might break after a version update.

Kubernetes provides YAML and JSON-formatted output options that are suitable for parsing or scripting.

YAML Output

The `-o yaml` option provides a YAML-formatted output that is parsable and still human-readable.

```
[user@host ~]$ oc get pods -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Pod
  metadata:
    annotations:
...object omitted...
```

> **NOTE**
>
> The reference documentation provides a more detailed introduction to YAML.

You can use any tool that can process YAML documents to filter the YAML output for your chosen field. For example, you can use the `yq` tool at https://mikefarah.gitbook.io/yq/ to process YAML and JSON files.

The `yq` processor uses a dot notation to separate field names in a query. The following example pipes the YAML output to the `yq` command to parse the `podIP` field.

```
[user@host ~]$ oc get pods -o yaml | yq '.items[0].status.podIP'
10.8.0.60
```

The `[0]` in the example specifies the first index in the items array.

> **NOTE**
>
> Another tool named `yq` is at https://kislyuk.github.io/yq/. The two `yq` tools are not compatible; commands that are designed for one of them do not work with the other.

JSON Output

Kubernetes uses the JSON format internally to process resource objects. Use the `-o json` option to view a resource in the JSON format.

```
[user@host ~]$ oc get pods -o json
{
    "apiVersion": "v1",
    "items": [
        {
            "apiVersion": "v1",
            "kind": "Pod",
            "metadata": {
                "annotations": {
```

You can use other tools to process JSON documents, such as the `jq` tool at https://jqlang.github.io/jq/. Similar to the `yq` processor, use the `jq` processor and dot notation on the fields to query specific information from the JSON-formatted output.

```
[user@host ~]$ oc get pods -o json | jq '.items[0].status.podIP'
"10.8.0.60"
```

Alternatively, the example might have used `.items[].status.podIP` for the query string. The empty brackets instruct the `jq` tool to query all items.

Custom Output

Kubernetes provides a custom output format that combines the convenience of extracting data via `jq` styled queries with a tabular output format. Use the `-o custom-columns` option with comma-separated *<column name> : <jq query string>* pairs.

```
[user@host ~]$ oc get pods \
-o custom-columns=PodName:".metadata.name",\
ContainerName:"spec.containers[].name",\
Phase:"status.phase",\
IP:"status.podIP",\
Ports:"spec.containers[].ports[].containerPort"
PodName                 ContainerName   Phase     IP          Ports
myapp-77fb5cd997-xplhz   myapp          Running   10.8.0.60   <none>
```

Kubernetes also supports the use of JSONPath expressions. JSONPath is a query language for JSON. JSONPath expressions refer to a JSON data structure; they filter and extract formatted fields from JSON objects.

In the following example, the JSONPath expression uses the `range` operator to iterate over the list of pods to extract the name of the pod, its IP address, and the assigned ports.

```
[user@host ~]$ oc get pods  \
-o jsonpath='{range .items[]}{"Pod Name: "}{.metadata.name}
{"IP: "}{.status.podIP}
{"Ports: "}{.spec.containers[].ports[].containerPort}{"\n"}{end}'
Pod Name: myapp-77fb5cd997-xplhz
IP: 10.8.0.60
Ports:
```

You can customize the format of the output with Go templates, which the Go programming language uses. Use the `-o go-template` option followed by a Go template, where Go expressions are inside double braces, `{{ }}`.

```
[user@host ~]$ oc get pods  \
-o go-template='{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}'
myapp-77fb5cd997-xplhz
```

REFERENCES

For more information, refer to the *OpenShift CLI (oc)* chapter in the Red Hat OpenShift Container Platform 4.18 *CLI Tools* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/cli_tools/index

For more information about custom columns, refer to the *oc get* section in the Red Hat OpenShift Container Platform 4.18 *CLI Tools* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/cli_tools/index#oc-get

For more detailed introduction to YAML, see the *YAML in a Nutshell* chapter in the Red Hat Enterprise Linux Atomic Host 7 *Getting Started with Kubernetes* documentation at https://docs.redhat.com/en/documentation/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_kubernetes/index#overview_3

Labels and selector details are available in the *Working with Kubernetes Objects* section of the Kubernetes Documentation - Labels and Selectors

Kubernetes object details are available in the *Understanding Kubernetes Objects* section of the Kubernetes Documentation - Understanding Kubernetes Objects

Command output details are available in the *Get* section of the Kubernetes Documentation - Getting Started

For more information on JSONPath operators and syntax, see the Kubernetes Documentation - JSONPath Support documentation.

For more information about Go templates and syntax, see the Go Templates documentation.