Cheat sheet

Red Hat OpenShift

Red Hat OpenShift is a unified platform to build, modernize, and deploy applications at scale. OpenShift uses the Kubernetes upstream project to provide a secure, robust, and extendable way to orchestrate applications. OpenShift works to further the access management and build and deploy services that the upstream Kubernetes project provides. Development teams can own and maintain their applications through production environments, while operations teams provide guidelines for developers to implement that application ownership in a multi-tenant environment.

This guide primarily focuses on the developer experience. However, it also details several administrator tasks. It provides a high-level listing of operations from the command-line interface, followed by a walkthrough of an example application build, deployment, and management methods.

Although the command list is not exhaustive, it covers most of the operations a developer should understand to manage an application's lifecycle.

Command overview

The `oc` tool, an essential component of Red Hat OpenShift, is a command-line interface (CLI) tool designed for managing Kubernetes clusters. It streamlines various operations, such as deploying applications, inspecting and managing cluster resources, and viewing logs.

Specifically tailored for OpenShift, an enterprise Kubernetes platform, the `oc` tool enhances the Kubernetes CLI `(kubectl)` by adding OpenShift-specific commands. This integration enables developers and system administrators to efficiently interact with OpenShift clusters, simplifying complex tasks and offering more control over the deployment and management of applications.

Simulate commands

Use the flag `–dry-run` to simulate the execution of an operation without actually performing it.

Version information

Use the flag `–dry-run` to simulate the execution of an operation without actually performing it.

```
oc version #shows the version number of the oc tool
```

Login and user management

```
oc login # Use —token to log in with an OAuth token or —username and —password to log in with your
OpenShift developer credentials.
```

```
oc logout # End the current session.
```

```
oc whoami # Show the current user context.
```

Project management

```
oc project # Shows the current project context, if a name is specified the project context is changed.
```

```
oc get projects # Show all projects the current login has access to.
```

```
oc status # Show overview of current project resources.
```

```
oc new-project # Create a new OpenShift project and change the context to it. Requires that the user
has the appropriate permissions to create projects.
```

Resource management

```
oc new-app # Create a new OpenShift application from an image, image stream, template or url. Use -S
[name] to search for images, image streams, and templates that match to [name].
```

```
oc new-build # Create a new OpenShift build configuration from an image, image stream or url.
```

```
oc label # Add/update/remove labels from an OpenShift resource.
```

```
oc annotate # Add/update/remove annotations from an OpenShift resource.
```

```
oc create # Create a new resource like a Pod or ConfigMap. Use -f [filename] to create from a manifest.
```

```
oc apply -f #Applies an object using a predefined yaml file
```

```
oc get # Retrieve a resource. Use -o for additional output options, and -f [filename] to retrieve
resources defined in a manifest.
```

```
oc get svc #Retrieves a list of services that have been defined. These services can be edited, similar
to other resources.
```

```
oc run # Create a pod from an image.
```

```
oc expose # Expose pods externally or internally. Use the pods/[name] argument to expose a pod to the
greater internet.
```

```
oc api-resources # List the supported api resources on the server.
```

```
oc explain # Get verbose details of an API resource.
```

```
oc replace # Replace an existing resource from a file name or stdin. Use -f - to replace resources
taken from stdin and -f [filename] to replace resources from a manifest.
```

```
oc delete # Delete a resource. Use -f [filename] to delete resources defined in a manifest.
```

```
oc edit # Modify a resource from a text editor. Use -f [filename] to modify resources defined in the
manifest.
```

```
oc describe # Retrieve a resource with details. Use -f [filename] to retrieve resource(s) defined in a
manifest.
```

Cluster management

```
oc adm # Administrative functions for an OpenShift cluster.
```

```
oc adm must-gather # Launch a new instance of a pod for gathering debug information. Use the –
[filename] argument to specify the output log path.
```

```
oc adm inspect # Collect debugging data for a given resource. Use -f [filename] to specify the
manifest describing the resource to collect information for.
```

```
oc adm policy # Manage role/scc to user/group bindings, as well as additional policy administration.
```

```
oc adm cordon/uncordon/drain # Unschedule/schedule/drain a node.
```

```
oc adm upgrade # Update the cluster. Use the –include-not-recommended argument to optionally install
upgrades not recommended for your cluster.
```

```
oc admin groups # Manage groups.
```

```
oc adm top # Show usage statistics of resources.
```

Additional resource management

```
oc patch # Update fields for a resource with JSON or YAML segments.
```

```
oc extract # Get ConfigMap(s) or Secret(s) and save the contents to disk.
```

```
oc set # Modify miscellaneous application resources.
```

```
oc set probe # Add a readiness/liveness probe for objects that have a pod template.
```

```
oc set volumes # Manage volume types on a pod configuration.
```

```
oc set build-hook # Update a build hook on a build config. Use the --script argument to specify a
shell script to run.
```

```
oc set env # Set configuration environment variables for objects that have a pod template.
```

```
oc set image # Update the image for objects that have a pod template.
```

```
oc set triggers # Set triggers for deployment configurations.
```

```
oc set serviceaccount # Update the service account for a resource.
```

```
oc set route-backends # Update the weight of a route to a service(s). Use –adjust [service]=percentage
to adjust the percentage of traffic from a route to a service by the specified percentage.
```

Operational commands

```
oc logs # Print the logs for a container in a pod.
```

```
oc rsh # Remote shell into a container.
```

```
oc rsync # Copy files between your computer's file system and a container.
```

```
oc exec # Execute a command in a container.
```

```
oc idle # Scale a resource to zero and disable any connected service.
```

Build and deploy commands

```
oc rollout # Manage deployments from deployment configuration.
```

```
oc rollout latest # Start a new deployment with the latest state.
```

```
oc rollout undo # Perform a rollback operation.
```

```
oc rollout history # View historical information for a Deployment configuration.
```

```
oc rollout status # Watch the status of a rollout.
```

```
oc tag # Tag existing images into image streams.
```

```
oc rollback # Rollback a pod or container to a previous configuration.
```

```
oc start-build # Start a new build from a build configuration.
```

```
oc cancel-build # Cancel a build in progress.
```

```
oc import-image # Pull in images and tags from an external container registry.
```

```
oc scale # Change the number of pod replicates.
```

```
oc debug #Deploys to a temporary container or pod, allowing you to inspect and diagnose issues
```

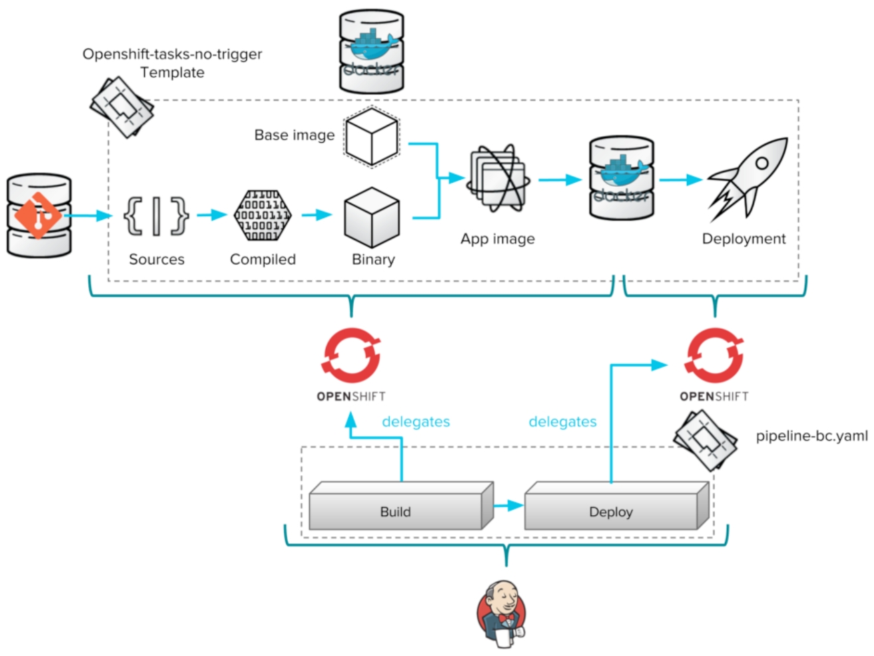
JsonPath

Using `jsonpath` with the `oc` tool in OpenShift allows for querying and extracting specific information from the JSON output of a command. This is particularly useful for parsing the complex JSON structures that are often returned by Kubernetes and OpenShift resources.

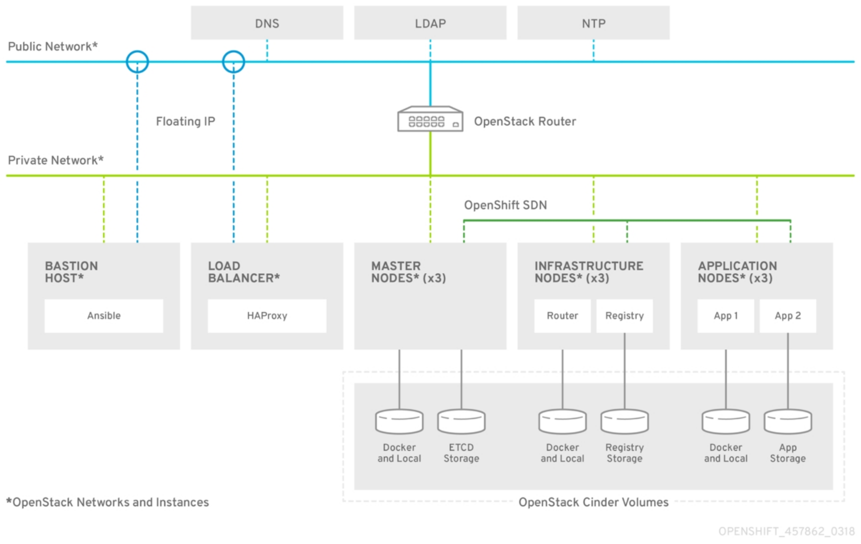
```
oc get [resource] -o jsonpath='{.jsonPathExpression}' #Here, [resource] is the type of resource you
are querying (like pods, services, etc.), and .jsonPathExpression is the jsonpath query expression you
want to use.
```

```
oc get pods -o jsonpath='{.items[*].metadata.name}' # This expression selects all items in the
returned JSON, navigates to their metadata, and then to the name field.
```

Simple build and deploy overview



Simple routing overview



Examples

Login

You can log in to the cluster to interact with OpenShift via the `oc` command:

```
$ oc login -u myuser https://openshift.example.com

Authentication required for https://openshift.example.com

Username: myuser

Password:
```

Note that leaving the `-p` option off of login prompts for the password. Additionally, you can verify your user context using the following command:

```
$ oc whoami

myuser
```

Create project

List the currently available projects you can view using the following command:

```
$ oc get projects
```

If this is your first time logging in and no one has added you to any existing projects, there shouldn't be any projects listed. Use the following command to create a project (allowed by the self-provisioner role to all authenticated users in the default OpenShift policy installation):

```
$ oc new-project myproject –display-name='My Project' –description='cool project owned by myuser'
```

Now, use `myproject` on the server https://openshift.example.com:443.

To build a new "Hello, World" example application on Ruby, you can add applications to this project with the `new-app` command. For example, try the following command.

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git
```

To view the project definition's specifics, output the full specs to YAML using the following command:

```
$ oc get project myproject -o yaml

apiVersion: project.openshift.io/v1

kind: Project

metadata:

    annotations:
```

```
    openshift.io/description: myproject
    openshift.io/display-name: myproject
    openshift.io/requester: myuser
    openshift.io/sa.scc.mcs: s0:c63,c17
    openshift.io/sa.scc.supplemental-groups: 1003940000/10000
    openshift.io/sa.scc.uid-range: 1003940000/10000
    toolchain.dev.openshift.com/last-applied-configuration:
    '{"apiVersion":"v1","kind":"Namespace","metadata":{"annotations":{"openshift.io/
    description":"myproject","openshift.io/display-name":"myproject","openshift.io/requester":"
    myuser"},"labels":{"modelmesh-enabled":"true","name":"myproject","opendatahub.io/
    dashboard":"true","toolchain.dev.openshift.com/provider":"codeready-toolchain","
    toolchain.dev.openshift.com/space":"myuser","toolchain.dev.openshift.com/type":"dev"},"
    name":"myproject"}}'
    toolchain.dev.openshift.com/last-applied-space-roles: '[{"templateRef":"base1ns-
    admin-88b275d-88b275d","usernames":["myuser"]}]'
    creationTimestamp: "2023-10-05T16:19:50Z"
    labels:
        kubernetes.io/metadata.name: myproject
        modelmesh-enabled: "true"
        name: myproject
        opendatahub.io/dashboard: "true"
        openshift-pipelines.tekton.dev/namespace-reconcile-version: 1.10.2
        pod-security.kubernetes.io/audit: baseline
        pod-security.kubernetes.io/audit-version: v1.24
        pod-security.kubernetes.io/warn: baseline
        pod-security.kubernetes.io/warn-version: v1.24
        toolchain.dev.openshift.com/provider: codeready-toolchain
        toolchain.dev.openshift.com/space: myuser
        toolchain.dev.openshift.com/templateref: base1ns-dev-3a789d0-3a789d0
        toolchain.dev.openshift.com/tier: base1ns
        toolchain.dev.openshift.com/type: dev
    name: myproject
    resourceVersion: "820595496"
    uid: c67ad567-fd9d-4555-9b6f-ae6366bfa6c1
    spec:
      finalizers:
      - kubernetes
    status:
      phase: Active
```

Add users to project

You can add additional users to your project by default using the following comman because self-provisioners get the admin role for any project they create.

```
$ oc adm policy add-role-to-user edit anotheruser

clusterrole.rbac.authorization.k8s.io/edit added: “anotheruser”
```

This approach lets `anotheruser` edit resources within the project but prevents them from managing the policy.

Create app from code and image

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git

–> Found container image 9244bc9 (3 weeks old) from registry.access.redhat.com for
“registry.access.redhat.com/ubi8/ruby-27”

    Ruby 2.7

    Ruby 2.7 available as a container is a base platform for building and running various Ruby 2.7
    applications and frameworks. Ruby is the interpreted scripting language for quick and easy object-
    oriented programming. It has many features to process text files and to do system management tasks
    (as in Perl). It is simple, straightforward, and extensible.

    Tags: builder, ruby, ruby27, ruby-27

    * An image stream tag will be created as “ruby-27:latest” that will track the source image

    * A Docker build using source code from https://github.com/openshift/ruby-hello-world.git will be
    created

    * The resulting image will be pushed to image stream tag “ruby-hello-world:latest”

    * Every time “ruby-27:latest” changes a new build will be triggered

    –> Creating resources …

    imagestream.image.openshift.io “ruby-27” created

    imagestream.image.openshift.io “ruby-hello-world” created

    buildconfig.build.openshift.io “ruby-hello-world” created

    deployment.apps “ruby-hello-world” created

    service “ruby-hello-world” created

    –> Success

    Build scheduled, use ‘oc logs -f buildconfig/ruby-hello-world’ to track its progress.

    Application is not exposed. You can expose services to the outside world by executing one or more

    of the commands below:

    ‘oc expose service/ruby-hello-world’

    Run ‘oc status’ to view your app.
```

The `new-app` command handles the majority of resource creation via template. Notice that it set up `deploymentconfig`, `buildconfig`, `service`, and `imagestream`.

Get resources

We can view the resources that the `new-app` command created, and the automatically created build and deploy resources. Notice that the `new-app` command automatically started a new build of the code. The `deployment config` watches successful builds to know when to roll out or deploy again.

To start viewing an application status, check the pods in your project using the following command:

```
$ oc get pods
```

The results are the following.

```
NAME                                                  READY     STATUS        RESTARTS     AGE
node-js-rest-api-openshift-example-1-build            0/1       Completed     0            10h
ruby-hello-world-1-build                              0/1       Completed     0            25m
ruby-hello-world-785bf649f7-484rk                     1/1       Running       0            24m
workspace7e300775f60c401d-94f6758f6-5qldj             2/2       Running       0            3h19m
```

This output shows that the Pod built successfully. Additionally, one ready and running Pod is deployed with the application.

The status command below shows similar results:

```
$ os status
```

The results are the following:

```
In project myproject on server https://openshift.example.com:443
svc/modelmesh-serving (headless) ports 8033, 8008, 8443, 2112

svc/node-js-rest-api-openshift-example-00001 - 172.30.166.225 ports 80->8012, 443->8112

svc/node-js-rest-api-openshift-example-00001-private - 172.30.4.101 ports 80->8012, 443->8112, 9090->http-autometric, 9091->http-usermetric, 8022, 8012

deployment/node-js-rest-api-openshift-example-00001-deployment deploys image-registry.openshift-image-registry.svc:5000/myproject/node-js-rest-api-openshift-example@sha256:f2dd0ceeaadadc576da5209ffc66c91017d327015ced21c4694156ea07dd8128,registry.redhat.io/openshift-serverless-1/serving-queue-

deployment #1 running for 10 hours

svc/ruby-hello-world - 172.30.189.176:8080

deployment/ruby-hello-world deploys istag/ruby-hello-world:latest <-
```

```
bc/ruby-hello-world docker builds https://github.com/openshift/ruby-hello-world.git on istag/
ruby-27:latest

deployment #2 running for 25 minutes - 1 pod

deployment #1 deployed 26 minutes ago

svc/workspace7e300775f60c401d-service - 172.30.18.174:4444

3 infos identified, use 'oc status –suggest' to see details.
```

Add a volume

To attach a volume to your Pods, you can use the `oc` set volume command:

```
$ oc set volume deployment/ruby-hello-world –add –mount-path=/mnt/emptydir

info: Generated volume name: volume-cfq8s

deployment.apps/ruby-hello-world volume updated
```

This example attached a simple `/mnt/emptydir` volume, though you can use the same command for Persistent Volumes. Also, notice that the deployment configuration has a `ConfigChange` trigger, so adding this volume starts a new deployment automatically.

Edit resource

Making a change to any OpenShift resource is straightforward. Use the following command to change the `/mnt/emptydir mount path above to /mnt/appdata`:

```
$ oc edit deployment/ruby-hello-world

# Please edit the object below. Lines beginning with a '#' will be ignored,

# and an empty file will abort the edit. If an error occurs while saving this file

# will be reopened with the relevant failures.

#

...

volumeMounts:

- mountPath:/mnt/emptydir /mnt/appdata

name: volume-7d1e8

...
```

Saving the file in your text editor updates the resource or reports errors if validation doesn't succeed. Note that this deployment config change initiates another app deployment.

Start build

To make a new build from the source, use the following command.

```
$ oc start-build ruby-hello-world

build.build.openshift.io/ruby-hello-world-2 started
```

Watch build

Watch the build logs using the following `oc` logs command and including `-f` for "follow":

```
$ oc edit deployment/ruby-hello-world
Edit cancelled, no changes made.
```

```
$ oc start-build ruby-hello-world
build.build.openshift.io/ruby-hello-world-2 started
```

```
$ oc logs -f deployment/ruby-hello-world
Found 2 pods, using pod/ruby-hello-world-5ddc8fc6bd-9vmdc
[2023-10-08 07:37:26] INFO  WEBrick 1.7.0
== Sinatra (v2.1.0) has taken the stage on 8080 for production with backup from WEBrick
[2023-10-08 07:37:26] INFO  WEBrick::HTTPServer#start: pid=7 port=8080
```

Watch deploy

Watch the overall deployment status via the following `oc` logs command.

```
$ oc logs -f buildconfig/ruby-hello-world
```

Additionally, you can view container logs with the following `oc` logs command.

```
$ oc logs ruby-hello-world-6bc699f648-wb6js

[2023-10-08 07:45:32] INFO  WEBrick 1.7.0

[2023-10-08 07:45:32] INFO  ruby 2.7.8 (2023-03-30) [x86_64-linux]

== Sinatra (v2.1.0) has taken the stage on 8080 for production with backup from WEBrick

[2023-10-08 07:45:32] INFO  WEBrick::HTTPServer#start: pid=7 port=8080
```

Debug

To create a temporary debug environment to diagnose issues, use the command:

```
$ oc debug [pod]/ruby-hello-world-5ddc8fc6bd-9vmdc
```

This command will create a new pod (`ruby-hello-world-5ddc8fc6bd-9vmdc-debug`, for instance) based on the existing pod but will not start the application containers. Instead, it will start a container with a shell so you can investigate the issue. Once the debug pod is running, you'll be given a command-line shell inside the pod.

When finished, you can destroy the debug environment by exiting with

```
$ exit
```

Use remote shell

Interacting directly with the container is straightforward with the following `oc rsh` command:

```
$ oc rsh ruby-hello-world-6bc699f648-wb6js

sh-4.4$ ls

Dockerfile  Gemfile  Gemfile.lock  README.md  Rakefile  app.rb  config  config.ru  db  models.rb
run.sh  test  views
```

Create route

```
$ oc expose service ruby-hello-world

route.route.openshift.io/ruby-hello-world exposed
```

With no other options defined, this command creates a route for your application using the default route naming (for example, `$appname-$projectname.openshift.example.com`).

Idle app

When you're done testing your application, you can idle the service to save resources. This command interacts with a Kubernetes Service to set the pod replicas to 0. Then, when something next accesses the service, the command automatically boots up the pods again.

```
$ oc idle ruby-hello-world

The service “myproject/ruby-hello-world” has been marked as idled

The service will unidle Deployment “myproject/ruby-hello-world” to 1 replicas once it receives traffic

Deployment “myproject/ruby-hello-world” has been idled
```

Delete app

When you're completely done with your application, you can delete resources within the project (or delete the project) to clean up, using the following command.

```
$ oc delete services -l app=ruby-hello-world
service "ruby-hello-world" deleted
```

```
$ oc delete all -l app=ruby-hello-world
deployment.apps "ruby-hello-world" deleted
buildconfig.build.openshift.io "ruby-hello-world" deleted
imagestream.image.openshift.io "ruby-27" deleted
imagestream.image.openshift.io "ruby-hello-world" deleted
route.route.openshift.io "ruby-hello-world" deleted
```

```
$ oc delete project myproject
Project "myproject" deleted
```