Project Request Template, Quotas, Limits

In the Application Management Basics lab, you dealt with the fundamental building blocks of OpenShift, all contained within a **Project**.

Out of the box, OpenShift does not restrict the quantity of objects or amount of resources (eg: CPU, memory) that they can consume within a **Project**. Further, users may create an unlimited number of **Projects**. In a world with no constraints, or in a POC-type environment, that would be fine. But reality calls for a little restraint.

Background: Project Request Template

When users invoke the `oc new-project` command, a new project request flow is initiated. During part of this this workflow, a default project **Template** is processed to create the newly requested project.

View the Default Project Request Template

To view the built-in default **Project Request Template**, execute the following command:

```
oc adm create-bootstrap-project-template -o yaml
```

Inspecting the the YAML output of this command, you will notice that there are various parameters associated with this **Template** displayed towards the bottom.

```
...
parameters:
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
- name: PROJECT_DESCRIPTION
- name: PROJECT_ADMIN_USER
- name: PROJECT_REQUESTING_USER
...


Next, let's take a look at the help output for the `oc new-project` command:
```

```
oc new-project -h
```

Notice there is a `--display-name` directive on `oc new-project`? This option corresponds to the `PROJECT_DISPLAYNAME` parameter we saw in the output of the default **Template**.

The `new-project` workflow involves the user providing information to fulfill the project request. OpenShift decides if this request should be allowed (for example, are users allowed to create **Projects**? Does this user have too many **Projects**?) and, if so, processes the **Template**.

If you look at the objects defined in the **Template**, you will notice that there's no mention of quota or limits. You'll change that now.

NOTE

Templates are a powerful tool that enable the user to easily create reusable sets of OpenShift objects with powerful parameterization. They can be used to quickly deploy more complicated and related components into OpenShift, and can be a useful part of your organization's software development lifecycle (SDLC). More information can be found in the template documentation. We will not go into more detail on **Templates** in these exercises.

Modify the Project Request Template

You won't actually have to make template changes in this lab — we've made them for you already. Use `cat`, `less`, or your favorite editor to view the modified **Project Request Template**:

```
cat {{ HOME_PATH }}/support/project_request_template.yaml
```

Please take note that there are two new sections added:**ResourceQuota** and **LimitRange**.

Background: ResourceQuota

The quota documentation provides a great description of **ResourceQuota**:

```
A resource quota, defined by a ResourceQuota object, provides constraints that
limit aggregate resource consumption per project. It can limit the quantity of
objects that can be created in a project by type, as well as the total amount of
compute resources and storage that may be consumed by resources in that
project.
```

In our case, we are setting a specific set of quota for CPU, memory, storage, volume claims, and **Pods**. Take a look at the `ResourceQuota` section from the `project_request_template.yaml` file:

```
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: ${PROJECT_NAME}-quota (1)
  spec:
    hard:
      pods: 10 (2)
      requests.cpu: 4000m (3)
      requests.memory: 8Gi (4)
      resourcequotas: 1
      requests.storage: 50Gi (5)
      persistentvolumeclaims: 5 (6)
      {{ CNS_STORAGECLASS }}.storageclass.storage.k8s.io/requests.storage: 25Gi (7)
      {{ CNS_BLOCK_STORAGECLASS }}.storageclass.storage.k8s.io/persistentvolumeclaims: 0 (8)
```

1. While only one quota can be defined in a **Project**, it still needs a unique name/id.
2. The total number of pods in a non-terminal state that can exist in the project.
3. CPUs are measured in "milicores". More information on how Kubernetes/OpenShift calculates cores can be found in the upstream documentation.
4. There is a system of both `limits` and `requests` that we will discuss more when we get to the **LimitRange** object.
5. Across all persistent volume claims in a project, the sum of storage requested cannot exceed this value.
6. The total number of persistent volume claims in a project.
7. This setting limits the amount of storage that can be provisioned using the `glusterfs-storage` **StorageClass**.
8. This setting limits the number of **PersistentVolumeClaims** for a **StorageClass** called `{{ CNS_BLOCK_STORAGECLASS }}`. A value of 0 means that creating **PersistentVolumeClaims** from this storage class is not allowed in this project.

For more details on the available quota options, refer back to the quota documentation.

Background: LimitRange

The limit range documentation provides some good background:

```
A limit range, defined by a LimitRange object, enumerates compute resource
constraints in a project at the pod, container, image, image stream, and
persistent volume claim level, and specifies the amount of resources that a pod,
container, image, image stream, or persistent volume claim can consume.
```

While the quota sets an upper bound on the total resource consumption within a project, the `LimitRange` generally applies to individual resources. For example, setting how much CPU an individual **Pod** or container can use.

Take a look at the sample `LimitRange` definition that we have provided in the `project_request_template.yaml` file:

```
- apiVersion: v1
  kind: LimitRange
  metadata:
    name: ${PROJECT_NAME}-limits
    creationTimestamp: null
  spec:
    limits:
      -
        type: Container
        max: (1)
          cpu: 4000m
          memory: 1024Mi
        min: (2)
          cpu: 10m
          memory: 5Mi
        default: (3)
          cpu: 4000m
          memory: 1024Mi
        defaultRequest: (4)
          cpu: 100m
          memory: 512Mi
```

The difference between requests and default limits is important, and is covered in the limit range documentation. But, generally speaking:

1. `max` is the highest value that may be specified for limits and requests
2. `min` is the lowest value that may be specified for limits and requests
3. `default` is the maximum amount (limit) that the container may consume, when nothing is specified
4. `defaultRequest` is the minimum amount that the container may consume, when nothing is specified

In addition to these topics, there are things like **Quality of Service Tiers** as well as a **Limit** : **Request** ratio. There is additionally more information in the compute resources section of the documentation.

For the sake of brevity, suffice it to say that there is a complex and powerful system of Quality of Service and resource management in OpenShift. Understanding the types of workloads that will be run in your cluster will be important to coming up with sensible values for all of these settings.

The settings we provide for you in these examples generally restrict projects to:

- A total CPU quota of 4 cores (`4000m`) where
  - Individual containers
    - must use 4 cores or less
    - cannot be defined with less than 10 milicores
    - will default to a request of 100 milicores (if not specified)
    - may burst up to a limit of 4 cores (if not specified)
- A total memory usage of 8 Gibibyte (8192 Megabytes) where
  - Individual containers
    - must use 1 Gi or less
    - cannot be defined with less than 5 Mi
    - will default to a request of 512 Mi
    - may burst up to a limit of 1024 Mi

- Total storage claims of 25 Gi or less
- A total number of 5 volume claims
- 10 or less **Pods**

In combination with quota, you can create very fine-grained controls, even across projects, for how users are allowed to request and utilize OpenShift's various resources.

> **NOTE**
>
> Remember that quotas and limits are applied at the **Project** level. **Users** may have access to multiple **Projects**, but quotas and limits do not apply directly to **Users**. If you want to apply one quota across multiple **Projects**, then you should look at the multi-project quota documentation. We will not cover multi-project quota in these exercises.

Installing the Project Request Template

With this background in place, let's go ahead and actually tell OpenShift to use this new **Project Request Template**.

Create the Template

As we discussed earlier, a **Template** is just another type of OpenShift object. The `oc` command provides a `create` function that will take YAML/JSON as input and simply instantiate the objects provided.

Go ahead and execute the following:

```
oc create -f {{ HOME_PATH }}/support/project_request_template.yaml -n openshift-config
```

This will create the **Template** object in the `openshift-config` **Project**. You can now see the **Templates** in the `openshift-config` project with the following:

```
oc get template -n openshift-config
```

You will see something like the following:

```
NAME              DESCRIPTION   PARAMETERS    OBJECTS
project-request                 5 (5 blank)   7
```

Setting the Default ProjectRequestTemplate

The default **projectRequestTemplate** is part of the OpenShift API Server configuration. This configuration is ultimately stored in a **ConfigMap** in the `openshift-apiserver` project. You can view the API Server configuration with the following command:

```
oc get cm config -n openshift-apiserver -o jsonpath --template="{.data.config\.yaml}" | jq
```

There is an OpenShift operator that looks at various **CustomResource** (CR) instances and ensures that the configurations they define are implemented in the cluster. In other words, the operator is ultimately responsible for creating/modifying the **ConfigMap**. You can see in the `jq` output that there is a `projectRequestMessage` but no `projectRequestTemplate` defined. There is currently no CR specifying anything, so the operator has configured the cluster with the "stock" settings. To add the default project request tempalate configuration, a CR needs to be created. The **CustomResource** will look like:

```
apiVersion: "config.openshift.io/v1"
kind: "Project"
metadata:
  name: "cluster"
  namespace: ""
spec:
  projectRequestMessage: ""
  projectRequestTemplate:
    name: "openshift-config/project-request"
```

Notice the **projectRequestTemplate** name matches the name of the template we created earlier in the `openshift-config` project.

The next step is to create this **CustomResource**. Once this **CR** is created, the OpenShift operator will notice the **CR**, and apply the configuration changes. To create the **CustomResource**, issue this command:

```
oc apply -f {{ HOME_PATH }}/support/cr_project_request.yaml -n openshift-config
```

Once this command is run, the OpenShift API Server configurations will be updated by the operator. This can be verified by viewing the implemented configuration:

```
oc get cm config -n openshift-apiserver -o jsonpath --template="{.data.config\.yaml}" | jq
```

Notice the new **projectConfig** section:

```
...
  "kind": "OpenShiftAPIServerConfig",
  "projectConfig": {
    "projectRequestMessage": "",
    "projectRequestTemplate": "openshift-config/project-request"
  },
...
```

Create a New Project

When creating a new project, you should see that a **Quota** and a **LimitRange** are created with it. First, create a new project called `template-test`:

```
oc new-project template-test
```

Then, use `describe` to look at some of this **Project's** details:

```
oc describe project template-test
```

The output will look something like:

```
Name:  template-test
Created: 7 seconds ago
Labels:  <none>
Annotations: openshift.io/description=
  openshift.io/display-name=
  openshift.io/requester=system:admin
  openshift.io/sa.scc.mcs=s0:c10,c0
  openshift.io/sa.scc.supplemental-groups=1000090000/10000
  openshift.io/sa.scc.uid-range=1000090000/10000
Display Name: <none>
Description: <none>
Status:  Active
Node Selector: <none>
Quota:
 Name:          template-test-quota
 Resource         Used Hard
 --------         ---- ----
 persistentvolumeclaims        0 5
 pods          0 10
 requests.cpu         0 4
 requests.memory         0 8Gi
 requests.storage        0 50Gi
 resourcequotas         0 1
Resource limits:
 Name:  template-test-limits
 Type  Resource Min Max Default Limit Limit/Request
 ----  -------- --- --- --- ----- -------------
 Container memory  5Mi 1Gi 1Gi 1Gi -
 Container cpu  10m 4 4 4 -
```

If you don't see the Quota and Resource limits sections, you may have been too quick. Remember that the operator takes a moment to do everything it needs to, so it's possible you created your project before the masters picked up the new configs. Go ahead and `oc delete project template-test` and then re-create it after a few moments.

You can also see that the **Quota** and **LimitRange** objects were created:

```
oc get quota -n template-test
```

You will see:

```
NAME                  CREATED AT
template-test-quota   2019-03-30T14:26:43Z
```

And:

```
oc get limitrange -n template-test
```

You will see:

```
NAME                   CREATED AT
template-test-limits   2018-10-24T19:19:40Z
```

NOTE Please make sure that the `project-request` template is created in the `openshift-config` project. Defining it in the OpenShift API server configurion without having the template in place will cause new projects to fail to create.

Clean Up

If you wish, you can deploy the application from the Application Management Basics lab again inside this `template-test` project to observe how the **Quota** and **LimitRange** are applied. If you do, be sure to look at the JSON/YAML output (`oc get … -o yaml`) for things like the **DeploymentConfig** and the **Pod**.

Before you continue, you may wish to delete the **Project** you just created:

```
oc delete project template-test
```

Last updated 2020-12-08 11:31:58 +0100