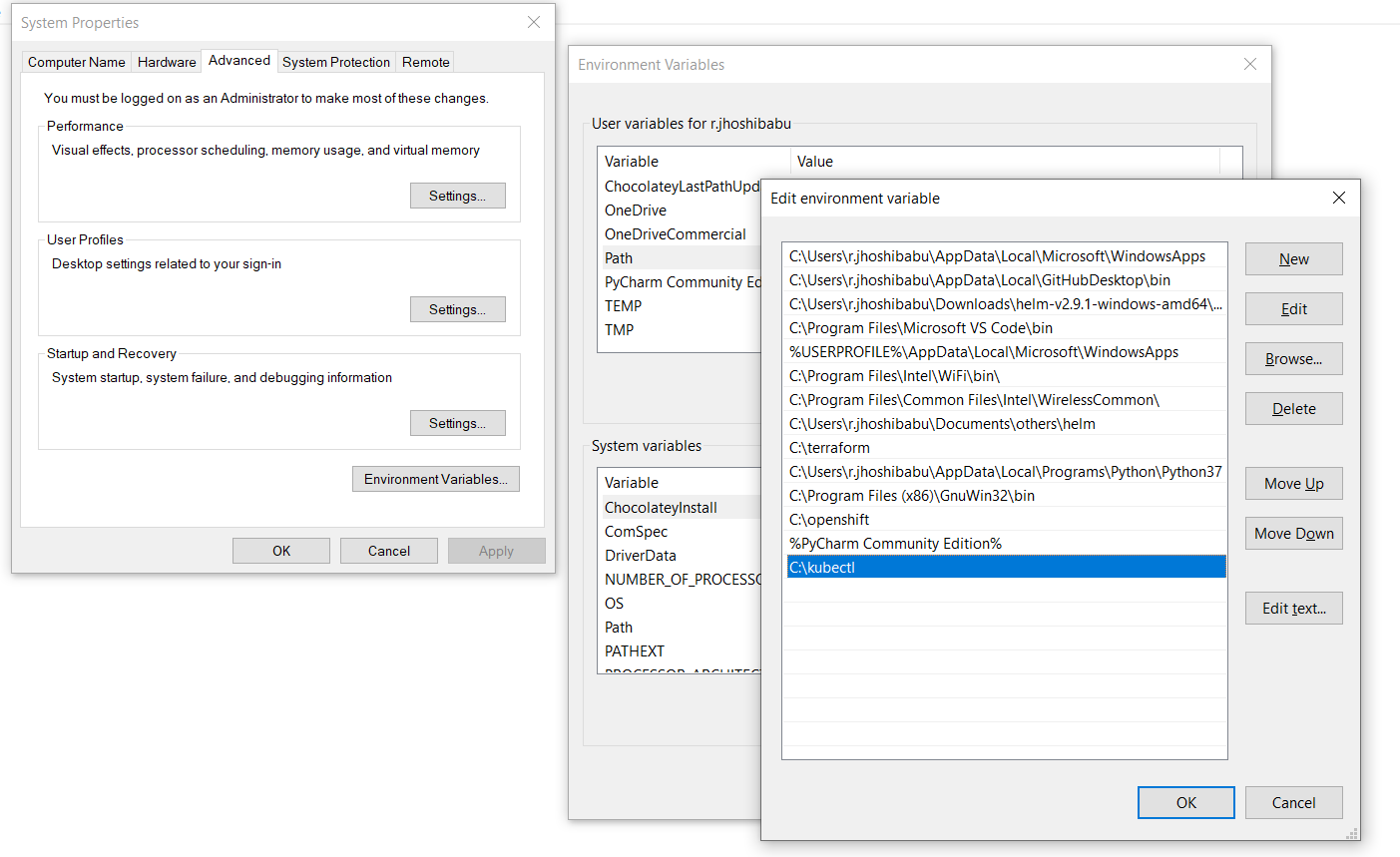
# **Lab 1: Install kubectl on Windows**

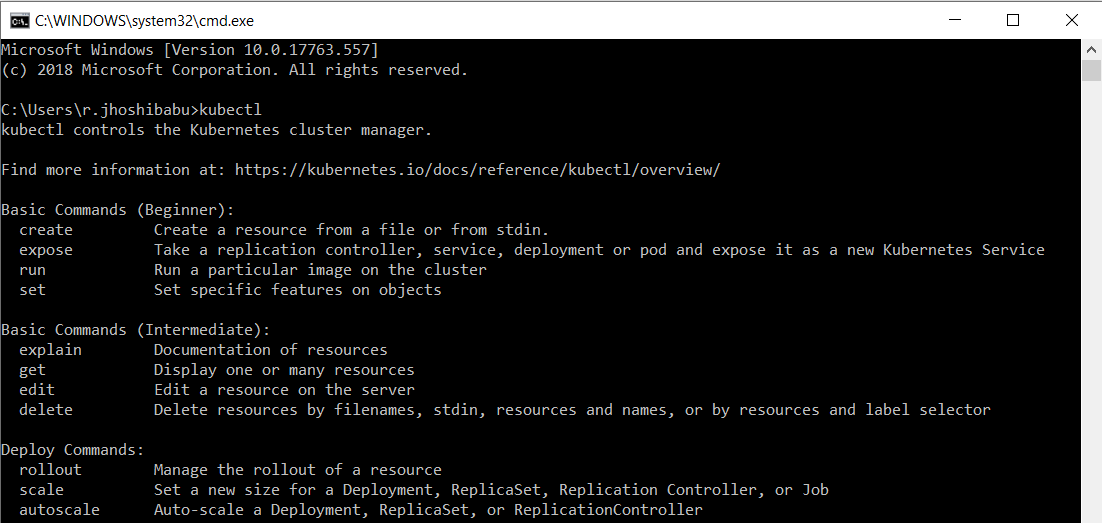
1. Download **kubectl.exe** tool from this [link](https://storage.googleapis.com/kubernetes-release/release/v1.15.0/bin/windows/amd64/kubectl.exe) and save the file in any folder on windows file system.

Eg.: Save file to “C:/kubectl/kubectl.exe”

1. Add the **kubectl.exe** folder location in path variable - “**Advanced System Settings** > **Advanced** > **Environment Variables** -> Path”. For example, if you have saved file to C:/kubectl then add this folder path to the path variable.



1. Open a command prompt and type ***kubectl*** and you should see all commands.



1. Create a .kube folder under the home profile using below command.

cd %USERPROFILE% && mkdir .kube

1. Copy the given kube **config** file into the .kube folder. This is used for connecting the Kubernetes cluster from your local system using kubectl command tool.
2. You can verify the connection of Kubernetes cluster by running below command and should able to see number of nodes.

Kubectl get nodes

# **Install kubectl on Linux**

1. Download the latest release with the command:

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

1. Make the kubectl binary executable.

chmod +x ./kubectl

1. Move the binary in to your PATH.

sudo mv ./kubectl /usr/local/bin/kubectl

1. Create a .kube folder under the home profile using below command.

mkdir $HOME/.kube && cd $HOME/.kube

1. Test to ensure the version you installed is up-to-date:

kubectl version

# **Lab 2: Deploying Kubernetes Dashboard**

The Dashboard UI is not deployed by default on Kubernetes cluster. To deploy it follow the below steps:

1. **Deploying Dashboard UI**

Create a certs directory under the user profile which is used to store the certificate files.

mkdir $HOME/certs && cd certs

To access Dashboard a valid certificate should be used. You can use self-signed certificates are generated and stored in-memory.

openssl req -new -newkey rsa:2048 -x509 -sha256 -days 365 -nodes -out dashboard.crt -keyout dashboard.key

Create a secret object named “*kubernetes-dashboard-certs*” to store custom certificates in “*kube-system*” namespace.

kubectl create secret generic kubernetes-dashboard-certs --from-file=$HOME/certs -n kube-system

Deploy the dashboard using below command

kubectl create -f <https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml>

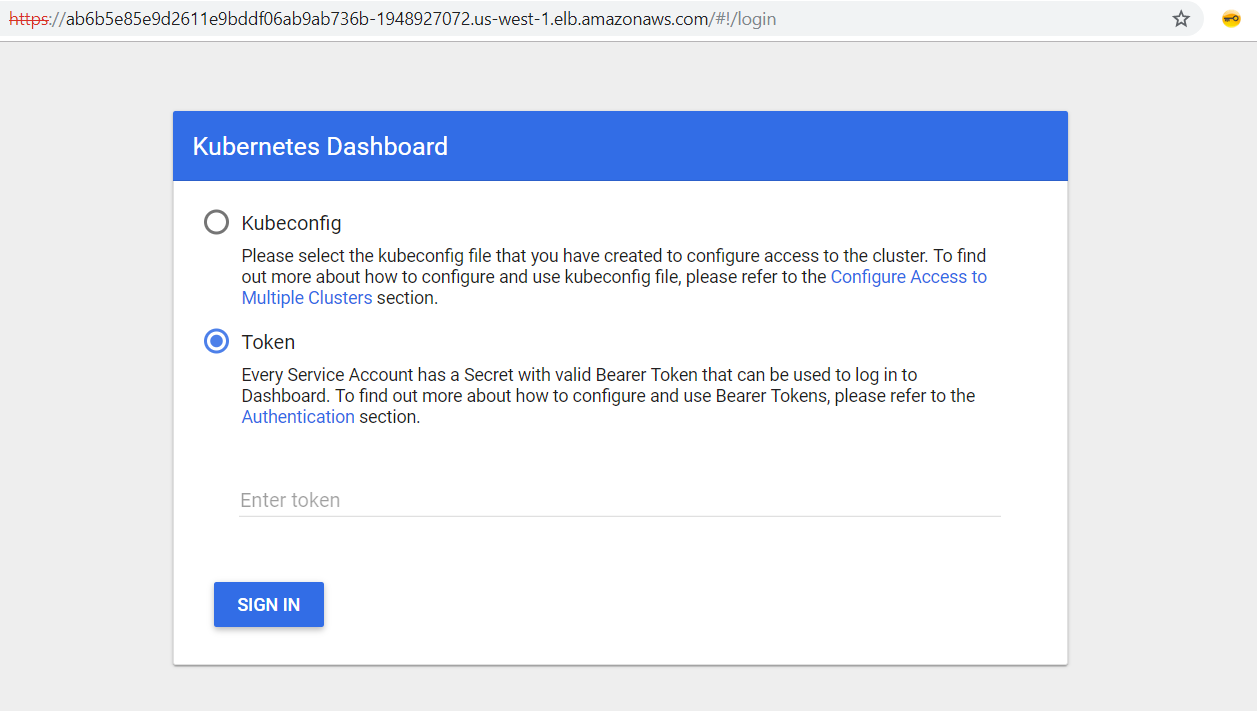
Edit the kubernetes-dashboard service and update the type to **LoadBalancer**.

kubectl edit svc kubernetes-dashboard -n kube-system

Type: LoadBalancer

To access the dashboard get the external domain url by running below command.

Kubectl get svc kubernetes-dashboard -n kube-system



1. **Authenticating Dashboard UI**

Create a service account for a dashboard in default namespace

kubectl create serviceaccount my-dashboard -n default

Add the cluster binding rules to your dashboard service account.

kubectl create clusterrolebinding my-dashboard --clusterrole=cluster-admin --serviceaccount=default:my-dashboard

Copy the secret token required for your dashboard login using below command.

kubectl get secret $(kubectl get serviceaccount my-dashboard -o jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 --decode

# **Lab 3: Deploying Application**

1. **Creating Deployment**

To create a deployment, copy the YAML formatted template shown below and save it locally as ***deployment.yaml*** file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{DEPLOYMENT\_NAME}}

labels:

app: {{DEPLOYMENT\_LABEL}}

spec:

replicas: 3

selector:

matchLabels:

app: {{DEPLOYMENT\_LABEL}}

template:

metadata:

labels:

app: {{DEPLOYMENT\_LABEL}}

spec:

containers:

- name: {{CONTAINER\_NAME}}

image: {{CONTAINER\_IMAGE}}

ports:

- containerPort: {{CONTAINER\_PORT}}

Example**:** *deployment.yaml*

apiVersion: apps/v1

kind: Deployment

metadata:

name: django

labels:

app: django

spec:

replicas: 3

selector:

matchLabels:

app: django

template:

metadata:

labels:

app: django

spec:

containers:

- name: django

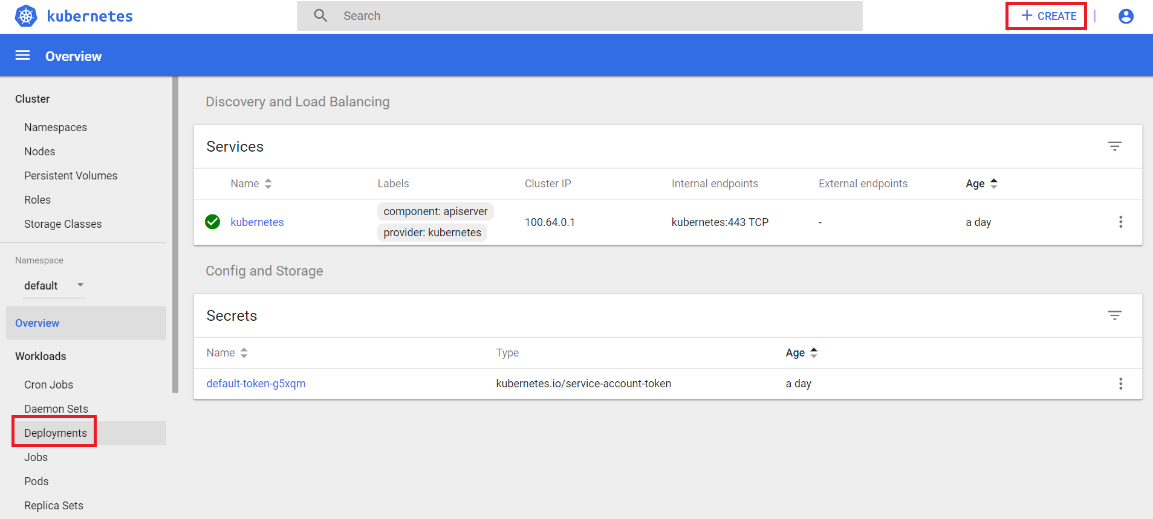
image: nexgtech/django

ports:

- containerPort: 8000

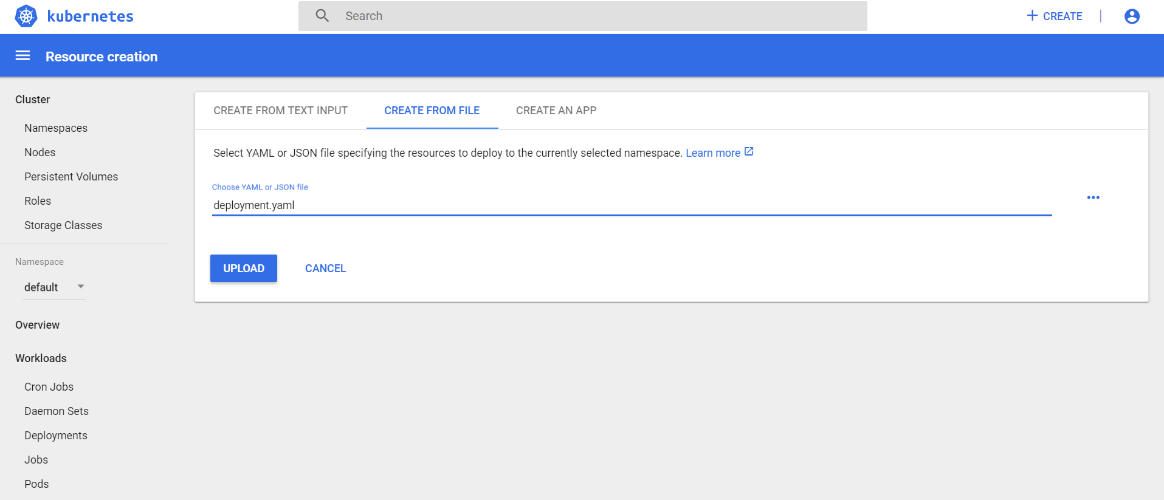
1. **Dashboard UI:**

On Kubernetes dashboard and navigate to **Workloads** > **Deployments** and Click **CREATE** in top-right corner.



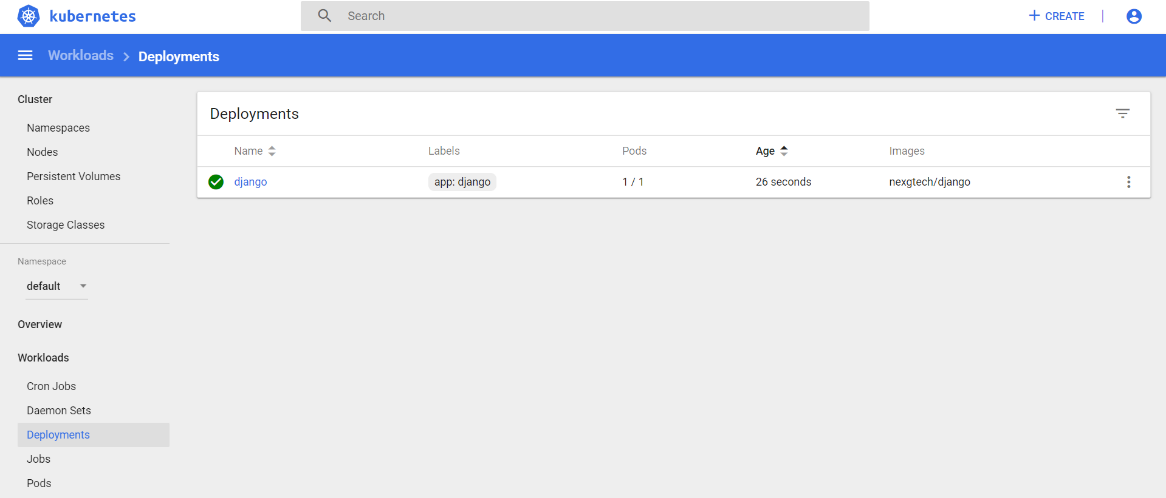
1. **Upload Template**

Select **CREATE FROM FILE** tab, upload the modified **deployment.yaml** file and click **UPLOAD**.



1. **Validate Deployment**

Navigate back to **Workloads** > **Deployments** to validate your deployment.



# **Lab 4: Creating Service**

1. **Creating service file**

***service.yaml*** file.

apiVersion: v1

kind: Service

metadata:

name: {{SERVICE\_NAME}}

spec:

selector:

app: {{DEPLOYMENT\_LABEL}}

ports:

- protocol: {{PROTOCOL}}

port: {{HOST\_PORT}}

targetPort: {{CONTAINER\_PORT}}

type: LoadBalancer

Example: *service.yaml*

apiVersion: v1

kind: Service

metadata:

name: django

spec:

selector:

app: django

ports:

- protocol: TCP

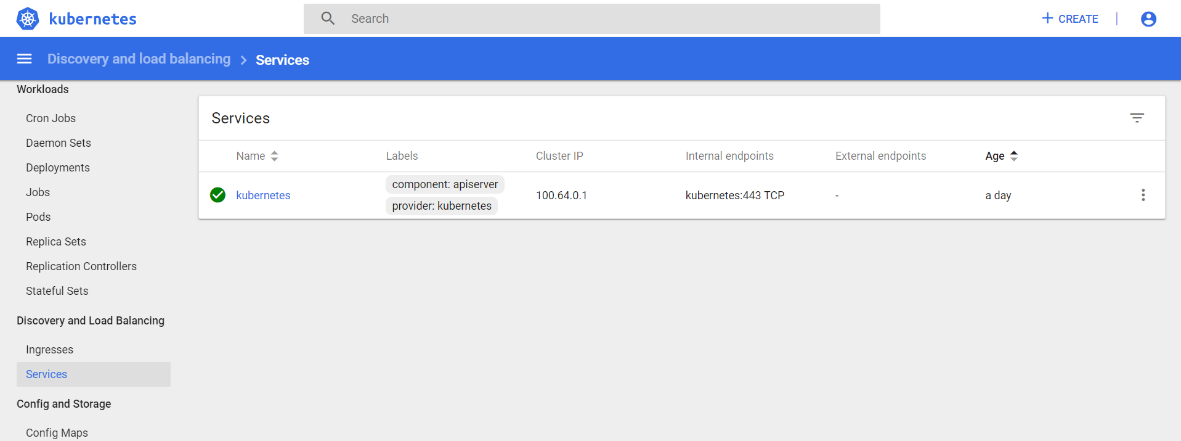
port: 80

targetPort: 8000

type: LoadBalancer

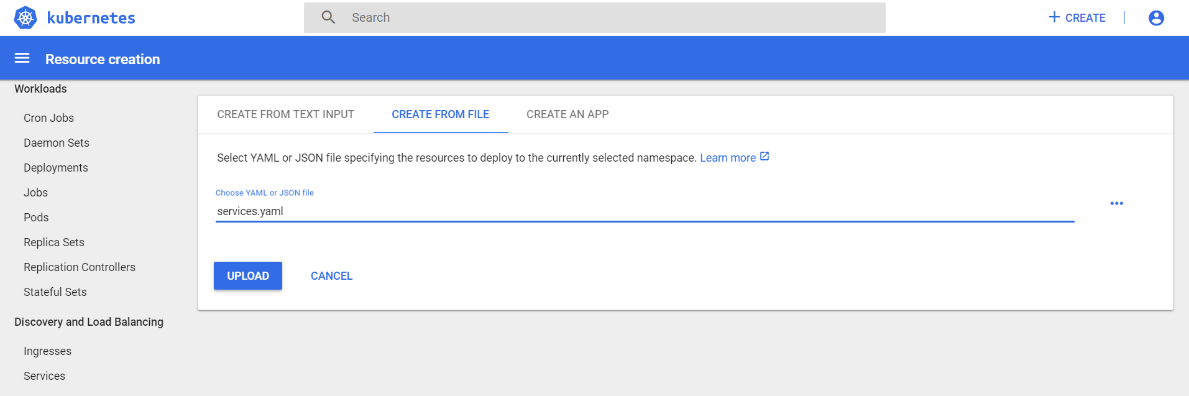
1. **Discovery and Load Balancing**

Navigate to **Discovery and Load Balancing** > **Service** and click **CREATE** in the top-right corner.



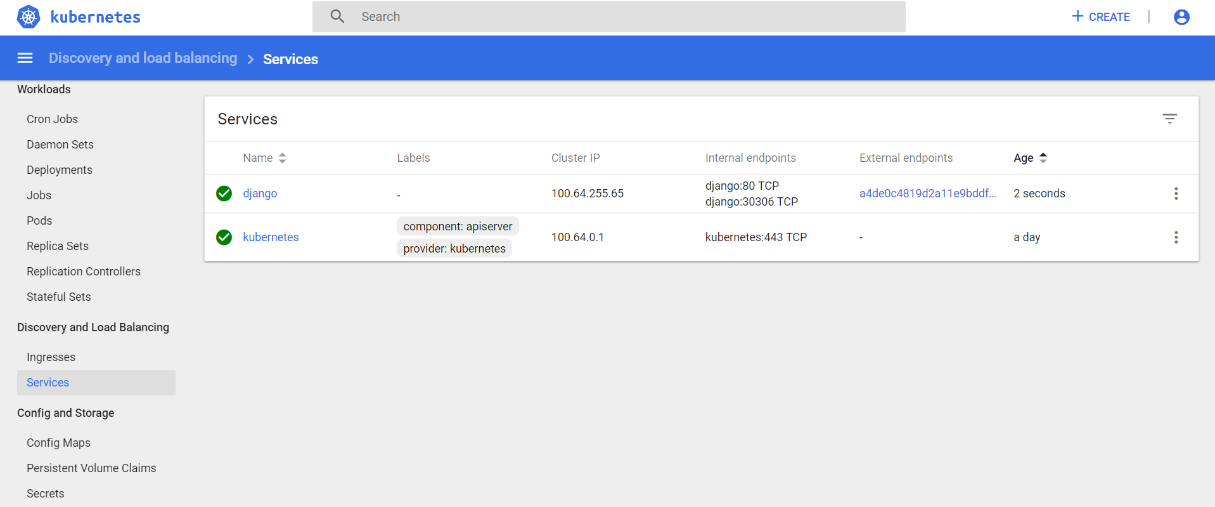
1. **Upload template**

Select **Uplaod a YAML or JSON file**, upload you **service.yaml** and click **UPLOAD**.



1. **Validate the Service**

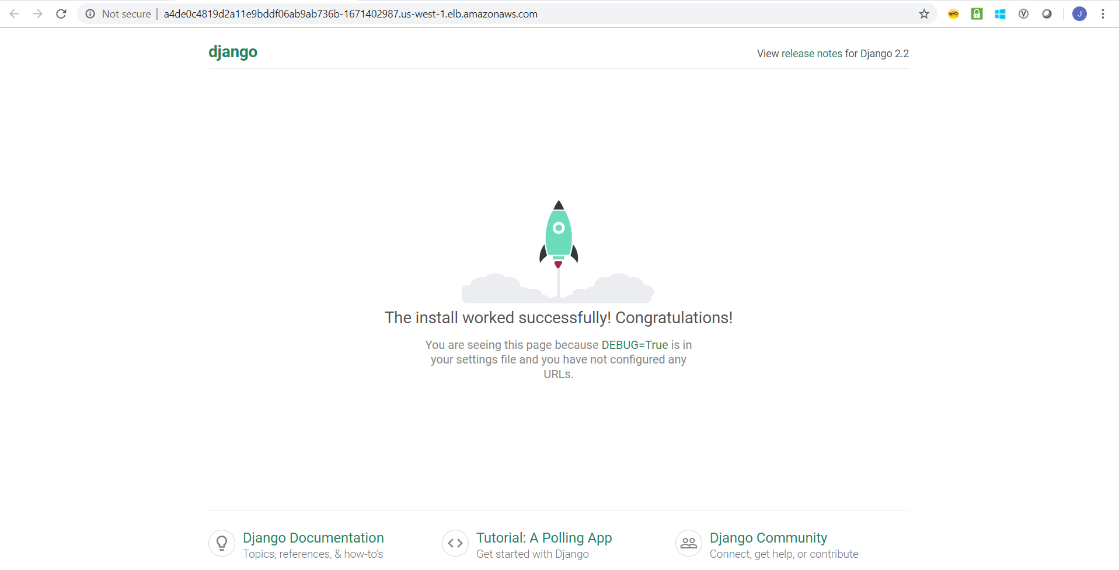
Navigate back to **Discovery and Load Balancing** > **Service** to validate you service. Wait for few seconds for the load balancer(External endpoint get updated) to be created.



**Note:** Refresh your page to check the progress. Once your **External endpoint** has changed from - to a DNS name of the ELB, your service has been created and exposed.

1. **Test Application**

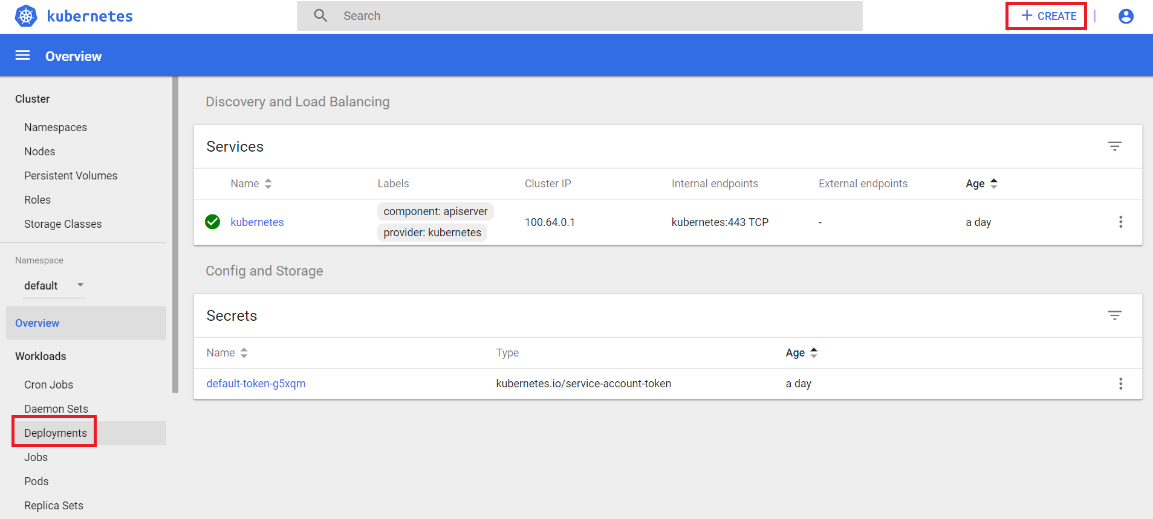
Validate your application is up and running. Click the name of the external endpoint to open your application in a new tab.



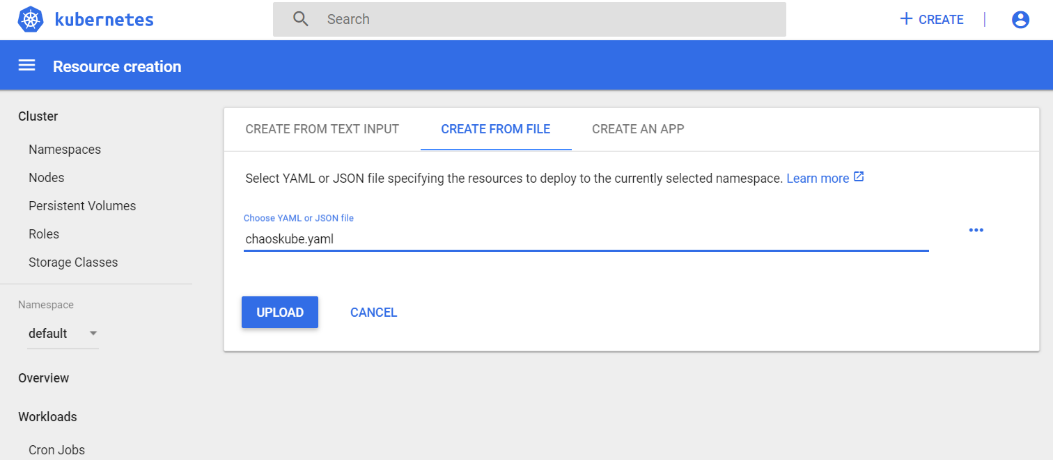
# **Lab 4: Deploying chaoskube**

Chaoskube periodically kills the random pods in the Kubernetes cluster. You can configure chaoskube to include or exclude pods based on labels, annotations, and namespaces, and to avoid certain time periods or dates.

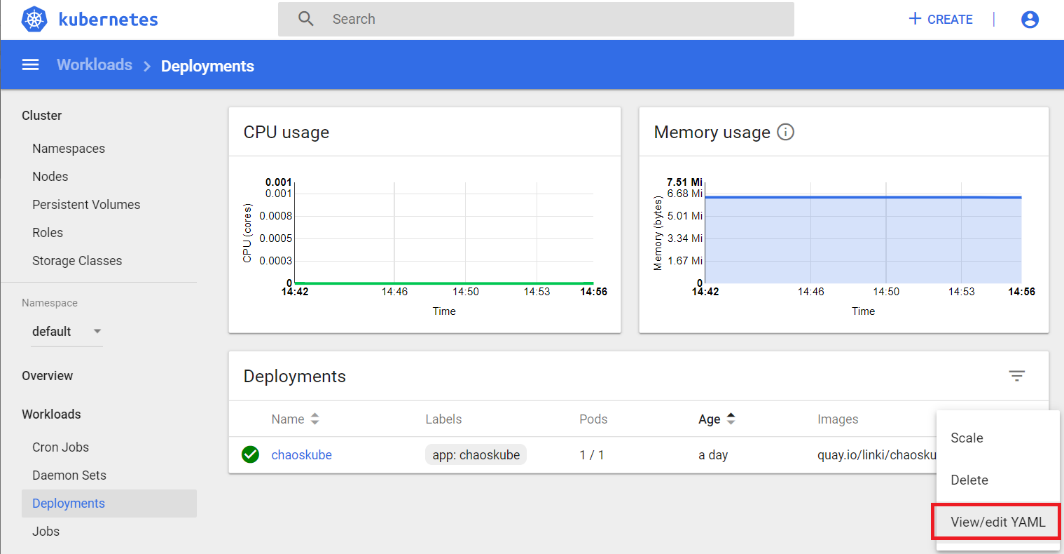
1. Deploy Chaos-kube application.
2. On Kubernetes dashboard and navigate to **Workloads** > **Deployments** and Click **CREATE** in top-right corner.



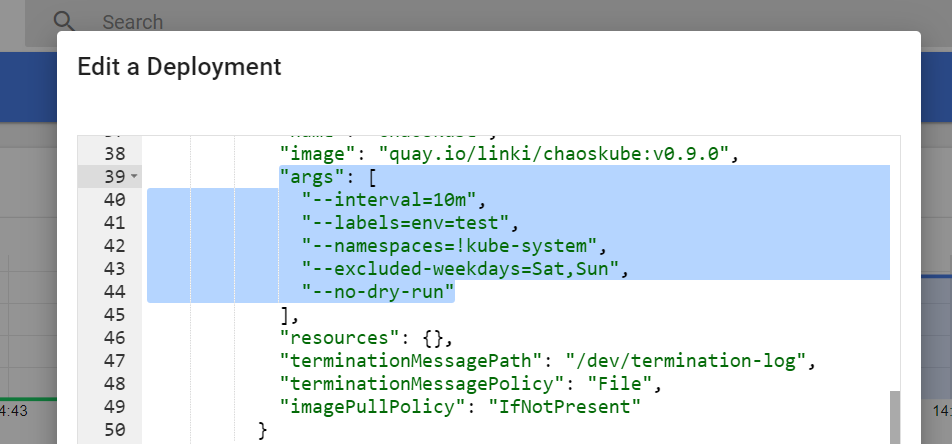
1. Select **Uplaod a YAML or JSON file**, upload you **chaoskube.yaml** from local and click **UPLOAD.**



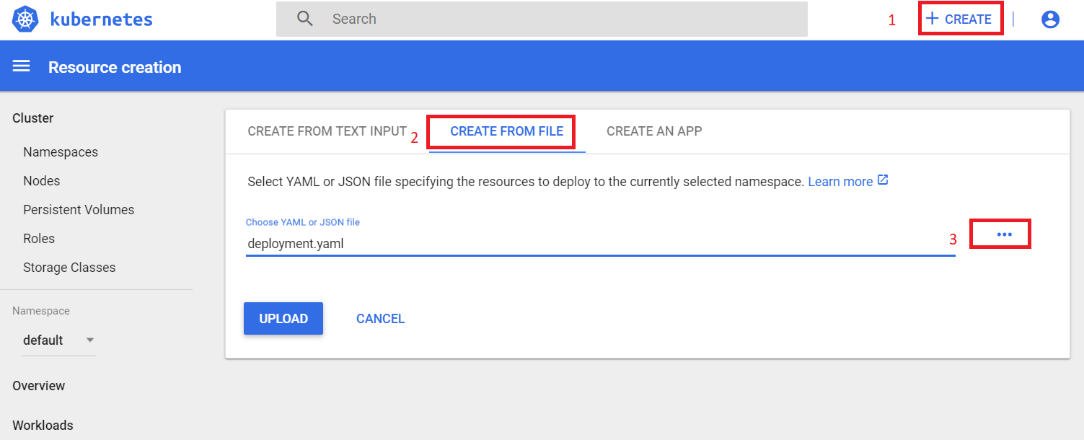
1. Go back to the **Deployments** under **Workloads**, select the action menu in the right corner of the chaoskube deployment and click **View/edit YAML**



1. Change the time interval parameter to **2 minutes (--interval=2m)**. By default, time interval to kill the pod is set to every 10 minutes in the deployment file.



1. Validating chaos-monkey by deploying sample application.
2. Create a new deployment by uploading the **deployment.yaml** from the Kubernetes dashboard**.**



**Note**: make sure deployment yaml contain label of ‘*env=test*’ under the ‘*spec.template.metadata.labels*’. so that chaoskube only target pods with label of ‘*env=test*’.

1. After deploying the sample application go to the **Workload** > **Pod,** and you can see three django pods running with different age, this is because every 2 minutes chaoskube randomly kills the pods.

