Pull Image from DockerHub Private Registry using Helm in Kubernetes

Updated: January 10, 2024 by Prasad Hole

In this article we are going to cover Pull Image from DockerHub Private Registry using Helm in Kubernetes.

Containerization has become a cornerstone in modern application development and deployment. Docker containers, for instance, encapsulate an application and its dependencies, ensuring consistency across various environments. To manage these containers efficiently, many organizations use container orchestration tools like Kubernetes, and to simplify deployment, Helm charts provide a powerful abstraction.

In this article, we'll explore how to pull images from a private registry using YAML and Helm. Private registries are essential for secure and controlled access to container images, and configuring Helm charts to fetch images from these private repositories is a common requirement.

Table of Contents

Prerequisites

- AWS Account with Ubuntu 22.04 LTS EC2 Instance
- A private container registry in DockerHub
- Minikube and kubectl, Helm Installed

Install Minikube and kubectl by following the official documentation for your operating system:

Minikube Installation Guide

Install Minikube on Ubuntu 22.04 LTS

- Helm Installed:

Install Helm by following the official documentation:

Helm Installation Guide

Push an Image in private registry

First create an image then push it in private Docker hub registry.

Here are some basic components to build a Docker image.

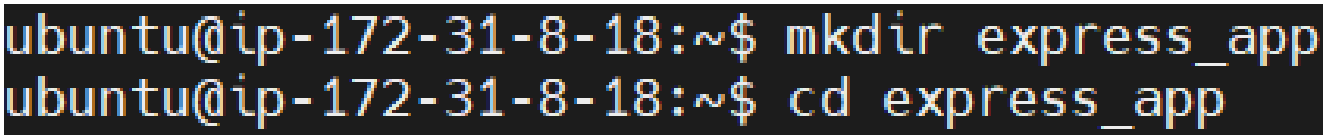
- Dockerfile

- Application Code
- Dependencies and Configuration Files

Step #1:Dockerizing a Node.js web app

First create a folder named **express_app** and move inside the folder using the following commands.

```
mkdir express_app
cd express_app
```



Step #2:Create the Node.js app

Then create a file named **package.json** which consist all files and dependencies required to describe the app.

```
nano package.json
```

add the following content into it.

```
{
  "name": "docker-example",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "nodemon": "^2.0.12"
```

```
    }
  }
```

```
{
  "name": "docker-example",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "nodemon": "^2.0.22"
  },
  "description": ""
}
```

save the modification using ctrl+x, shift+y and Enter.

install npm (Node Package Manager)

```
sudo apt install npm
```

Now, initialize the node project using the following command

```
npm init
```

```
ubuntu@ip-172-31-8-18:~/express_app$ npm init
```

Output:

```
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (docker-example)
version: (1.0.0)
description:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/ubuntu/express_app/package.json:

{
  "name": "docker-example",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "nodemon": "^2.0.12"
  },
  "description": ""
}


Is this OK? (yes)
```

The package.json file will be added. It holds information about our projects like scripts, files, dependencies, and versions. It will ask for name, version and many other things you can just set it to default by pressing Enter.

Then install the Express library and add it to the package.json file

```
npm install --save express
```

```
ubuntu@ip-172-31-8-18:~/express_app$ npm install --save express
```

Output:

```
added 94 packages, and audited 95 packages in 4s

14 packages are looking for funding
  run `npm fund` for details

3 moderate severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
ubuntu@ip-172-31-8-18:~/express_app$ npm install --save nodemon

up to date, audited 95 packages in 1s

14 packages are looking for funding
  run `npm fund` for details

3 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

Now install a tool called nodemon. It will automatically restarts the node application when it detects any changes.

```
npm install --save nodemon
```

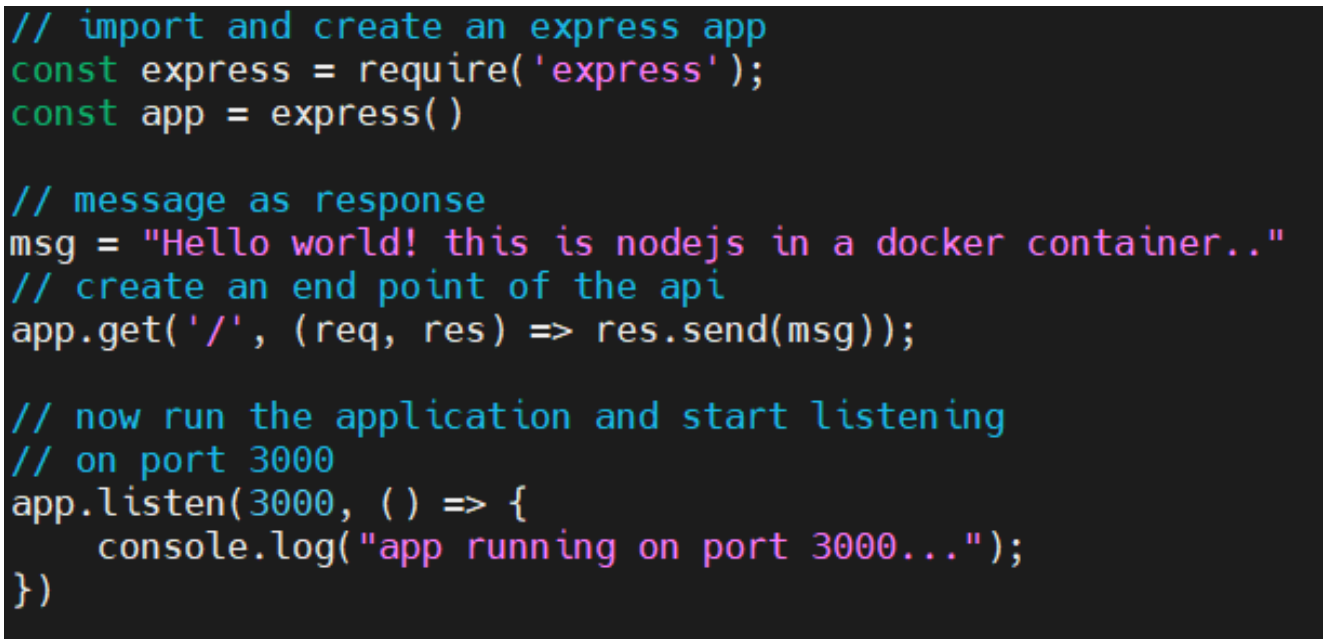
Step #3:Create app.js file for Nodejs app

Then, create a app.js file named app.js that defines a web app using the Express.js framework.

```
nano app.js
```

add the following content into it.

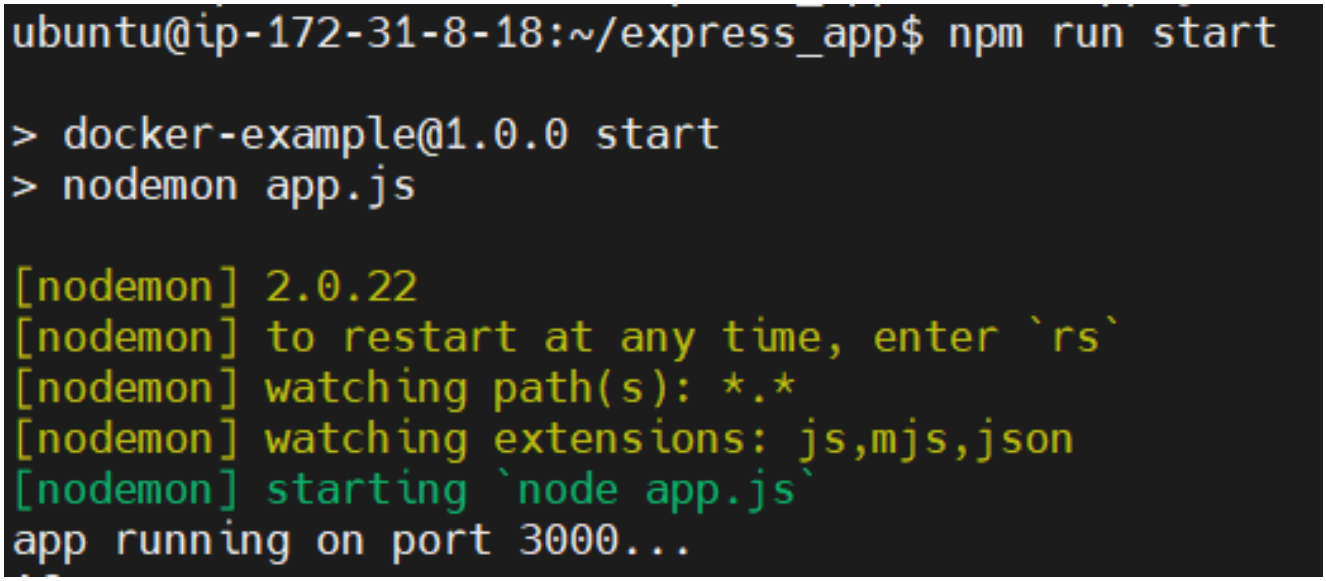
```
// import and create an express app
const express = require('express');
const app = express()

// message as response
msg = "Hello world! this is nodejs in a docker container.."
// create an end point of the api
app.get('/', (req, res) => res.send(msg));

// now run the application and start listening
// on port 3000
app.listen(3000, () => {
    console.log("app running on port 3000...");
})
```



save the modification using ctrl+x, shift+y and Enter.

After this we can run the application on our local system.

```
npm run start
```



Step #4:Create a Dockerfile For Nodejs app

Now to Dockerize the application we will create an image and for that we will create a Dockerfile which holds the information about image that will run the application.

```
nano Dockerfile
```

```
FROM node:latest
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
CMD ["npm", "start"]
EXPOSE 3000
```

```
FROM node:latest

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

CMD ["npm", "start"]

EXPOSE 3000
```

save the modification using ctrl+x, shift+y and Enter.

Step #5:Building an Image for Nodejs app

Now with the help of Dockerfile build an image using following command.

```
docker build -t nodeapp .
```

```
ubuntu@ip-172-31-8-18:~/express_app$ docker build -t nodeapp .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.791MB
Step 1/7 : FROM node:latest
latest: Pulling from library/node
bc0734b949dc: Pull complete
b5de22c0f5cd: Pull complete
917ee5330e73: Pull complete
b43bd898d5fb: Pull complete
9b5852c9e7e7: Pull complete
5b6382a075a3: Pull complete
773b85e76785: Pull complete
004ced7a943e: Pull complete
Digest: sha256:73a9c498369c6e6f864359979c8f4895f28323c07411605e6c870d696a0143fa
Status: Downloaded newer image for node:latest
 ---> b678102cb2bf
Step 2/7 : WORKDIR /app
 ---> Running in 8a20566edcdb
Removing intermediate container 8a20566edcdb
 ---> d84a03f6d9fd
Step 3/7 : COPY package.json /app
 ---> 0a6fc36f4c3e
Step 4/7 : RUN npm install
 ---> Running in 4166c308df48

added 94 packages, and audited 95 packages in 4s

14 packages are looking for funding
  run `npm fund` for details

3 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
npm notice
npm notice New patch version of npm available! 10.2.4 -> 10.2.5
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v10.2.5>
npm notice Run `npm install -g npm@10.2.5` to update!
npm notice
Removing intermediate container 4166c308df48
 ---> 64652a82ba11
Step 5/7 : COPY . /app
 ---> 49bbd623f775
Step 6/7 : CMD ["npm", "start"]
 ---> Running in f9699beb59f5
Removing intermediate container f9699beb59f5
 ---> f6c1191d843c
Step 7/7 : EXPOSE 3000
 ---> Running in bc33add7c22c
Removing intermediate container bc33add7c22c
 ---> 36b50b9653ff
Successfully built 36b50b9653ff
Successfully tagged nodeapp:latest
```

confirm the image is created by running following command.

```
docker images
```

Now run the image as a container using following command

```
docker run -p 3000:3000 nodeapp
```

```
ubuntu@ip-172-31-8-18:~/express_app$ docker run -p 3000:3000 nodeapp

> docker-example@1.0.0 start
> nodemon app.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
app running on port 3000...
```

Step #6:Push the image in private registry

Login to the docker

```
docker login
```

Output:

```
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: prasadhole
Password:
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

it will ask for username and password

then tag the image using following command.

```
docker tag nodeapp:latest prasadhole/nodeapp:latest
```

```
ubuntu@ip-172-31-8-18:~/express_app$ docker tag nodeapp:latest prasadhole/nodeapp:latest
```

then push the image to your private registry using following command

```
docker push prasadhole/nodeapp:latest
```

```
ubuntu@ip-172-31-8-18:~/express_app$ docker push prasadhole/nodeapp:latest
The push refers to repository [docker.io/prasadhole/nodeapp]
63e662b71f0e: Pushed
2b5defb7e1f8: Pushed
4283dbe6528b: Pushed
dcee60e2872d: Pushed
a198729ad0b3: Mounted from prasadhole/docker-container-nodejs
2bf51e018fe1: Mounted from prasadhole/docker-container-nodejs
453f42978239: Mounted from prasadhole/docker-container-nodejs
f300d546989d: Mounted from prasadhole/docker-container-nodejs
ac7146fb6cf5: Mounted from prasadhole/docker-container-nodejs
209de9f22f2f: Mounted from prasadhole/docker-container-nodejs
777ac9f3cbb2: Mounted from prasadhole/docker-container-nodejs
ae134c61b154: Mounted from prasadhole/docker-container-nodejs
latest: digest: sha256:d28348ae751904d9c7094cb02eee1bec1d75cd38326df82dda46249f6e44ae61 size: 2838
```

now image is pushed to the private registry named **nodeapp**.

Pull the image from private registry using helm file

Step #1:Create a secret for docker-registry

Create a secret Docker registry

```
kubectl create secret docker-registry nodeapp \
    --docker-server=https://index.docker.io/v1/ \
    --docker-username=prasadhole \
    --docker-password=Prasad@2002
```

```
ubuntu@ip-172-31-8-18:~$ kubectl create secret docker-registry nodeapp \
    --docker-server=https://index.docker.io/v1/ \
    --docker-username=prasadhole \
    --docker-password=Prasad@2002
secret/nodeapp created
```

Step #2:Create a helm chart for Nodejs app

Create a helm chart

```
helm create nodejs
```

```
ubuntu@ip-172-31-8-18:~$ helm create nodejs
Creating nodejs
```

If your Docker image is hosted on Docker Hub and your Docker Hub credentials are needed to pull the image during deployments in Kubernetes, you can create a secret with your Docker Hub credentials using the following `kubectl create secret docker-registry` command.

now open the directory

```
cd nodejs
```

then open the values.yaml file and modify it as shown below

```
nano values.yaml
```

```
# Default values for nodejs.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

image:
  repository: prasadhole/nodeapp
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "latest"

imagePullSecrets:
 - name: nodeapp
nameOverride: ""
fullnameOverride: ""
```

```
serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Automatically mount a ServiceAccount's API credentials?
  automount: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname
template
  name: ""

podAnnotations: {}
podLabels: {}

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

service:
  type: NodePort
 port: 3000

ingress:
  enabled: false
  className: ""
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths:
        - path: /
          pathType: ImplementationSpecific
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local

resources: {}
  # We usually recommend not to specify default resources and to leave
this as a conscious
  # choice for the user. This also increases chances charts run on
environments with little
  # resources, such as Minikube. If you do want to specify resources,
uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after
'resources:'.
  # limits:
  #   cpu: 100m
```

```
  #   memory: 128Mi
  # requests:
  #   cpu: 100m
  #   memory: 128Mi

autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
  # targetMemoryUtilizationPercentage: 80

# Additional volumes on the output Deployment definition.
volumes: []
# - name: foo
#   secret:
#     secretName: mysecret
#     optional: false

# Additional volumeMounts on the output Deployment definition.
volumeMounts: []
# - name: foo
#   mountPath: "/etc/foo"
#   readOnly: true

nodeSelector: {}

tolerations: []

affinity: {}
```

```
replicaCount: 1

image:
  repository: prasadhole/nodeapp
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "latest"

imagePullSecrets:
  - name: nodeapp
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Automatically mount a ServiceAccount's API credentials?
  automount: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""

podAnnotations: {}
podLabels: {}

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

service:
  type: NodePort
  port: 3000

ingress:
  enabled: false
  className: ""
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
```

save the modification using ctrl+x, shift+y and Enter.

exit the directory

```
cd
```

Step #3:Install the chart to Pull the image

now install the chart

```
helm install mynodeapp nodejs
```

```
ubuntu@ip-172-31-8-18:~$ helm install mynodeapp nodejs \
    --set imagePullSecrets[0].name=nodeapp
NAME: mynodeapp
LAST DEPLOYED: Tue Jan  2 11:31:07 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services mynodeapp-nodejs)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
```

run the kubectl command check the pod is running or not

```
kubectl get pods
```

```
ubuntu@ip-172-31-8-18:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mynodeapp-nodejs-b74b7db44-c6rsr    1/1     Running   0          25s
```

run the following command to check if image is pulled or not.

```
kubectl pod describe mynodeapp-nodejs-b74b7db44-c6rsr
```

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  62s   default-scheduler  Successfully assigned default/mynodeapp-nodejs-b74b7db44-jhprh to minikube
  Normal  Pulling    61s   kubelet            Pulling image "prasadhole/nodeapp:latest"
  Normal  Pulled     28s   kubelet            Successfully pulled image "prasadhole/nodeapp:latest" in 32.971s (32.971s including waiting)
  Normal  Created    27s   kubelet            Created container nodejs
  Normal  Started    27s   kubelet            Started container nodejs
```

Step #4:Run the Nodejs app on browser

For checking services, run following command

```
kubectl get svc
```

```
ubuntu@ip-172-31-8-18:~$ kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes        ClusterIP   10.96.0.1        <none>        443/TCP          5h46m
mynodeapp-nodejs  NodePort    10.107.102.227   <none>        3000:32325/TCP   76s
```

For accessing the Nodejs application on browser use following command.

```
kubectl port-forward --address 0.0.0.0 svc/mynodeapp-nodejs 3000:3000
```

```
ubuntu@ip-172-31-8-18:~$ kubectl port-forward --address 0.0.0.0 svc/mynodeapp-nodejs 3000:3000
Forwarding from 0.0.0.0:3000 -> 3000
```

This command is used to forward traffic from port 3000 on your local machine to port 3000 on the specified service **mynodeapp-nodejs**. This can be useful for accessing a service running in your Kubernetes cluster from your local machine.

To access the application on Browser write the ip address:port number in url.

ip address is the public ip address of your **Minikube EC2 instance** created on **AWS** and port number which is **3000** which we have used in forwarding **nodejs** pod.

" *Hello world! this is nodejs in a docker container..* " message will be displayed as follows.

Conclusion:

Pulling images from a private registry using YAML and Helm involves creating a Kubernetes Secret for authentication and updating Helm chart values. This ensures that your Kubernetes deployments can securely access container images from private repositories, providing a robust and scalable solution for containerized applications.

Related Articles:

How to Use Environment Variables in Helm Chart

About Prasad Hole

Leave a Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Search

Monitor Docker Containers with Prometheus and Grafana

Cloudways Autonomous-Kubernetes Hosting for High Traffic Site

4 Types of Elastic Load Balancer in AWS

Deploy Application to AWS Elastic Beanstalk

Pull Image from DockerHub Private Registry using Helm in Kubernetes

Terraform Cloud Sentinel Policy and Remote Backends

How to Create Amazon ECS Cluster

About DevOpsHint

DevOpsHint is a Community site where you can find about How to Guides, Articles and Troubleshooting Tips for Various current DevOps,GitOps,DevSecOps,SRE Tools and Resources.

DevOps Resources

Free DevOps Resources

Consulting and Support

Terraform with AWS Consulting and Job Support

Prometheus Consulting and Job Support

Site Links

About Us

Contact Us

Privacy Policy

Terms and Conditions