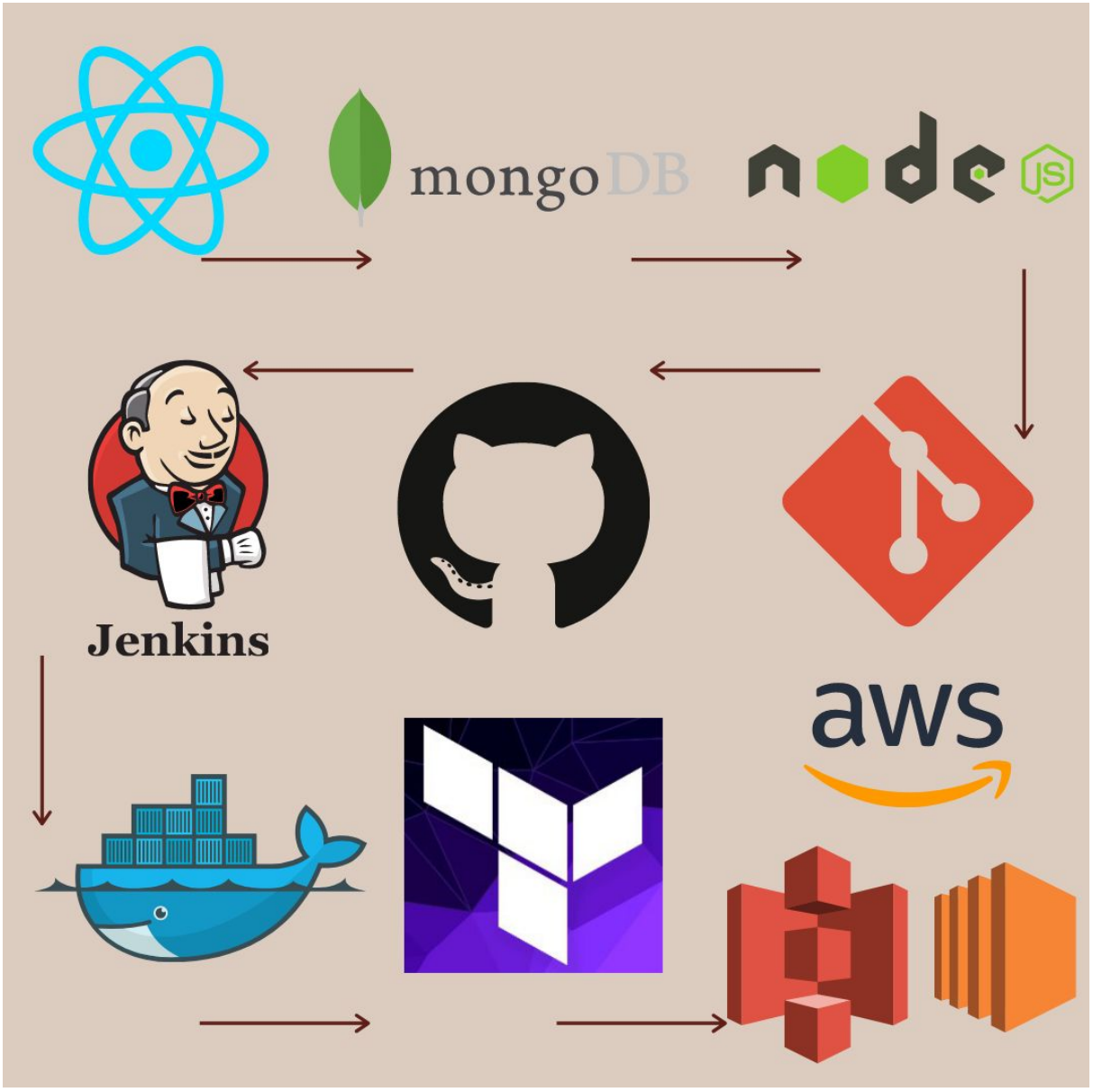
Hosting a containerized MERN Application on AWS Fargate



Tired of running your amazing MERN Stack applications locally? Let's use this template DevOps Stack to automate the hosting of your MERN Application on AWS Cloud

The Pipeline and Technology Stack

1. **Web Application Stack: MERN Stack**

- Frontend- React
- Backend- Mongo DB
- Package Management- Node JS
- API Management: Express

1. **Version Control:** Git with GitHub repository hosting

2. **Continuous Integration:** Jenkins
3. **Container Engine:** Docker
4. **Infrastructure Automation:** HashiCorp Terraform
5. **Cloud Services:** Amazon Web Services- AWS Fargate

The project is hosted on GitHub on this link.

The Procedure

Step 1) Building the MERN Application

The main website is based on the amazing tutorial created by Brad Traversy.

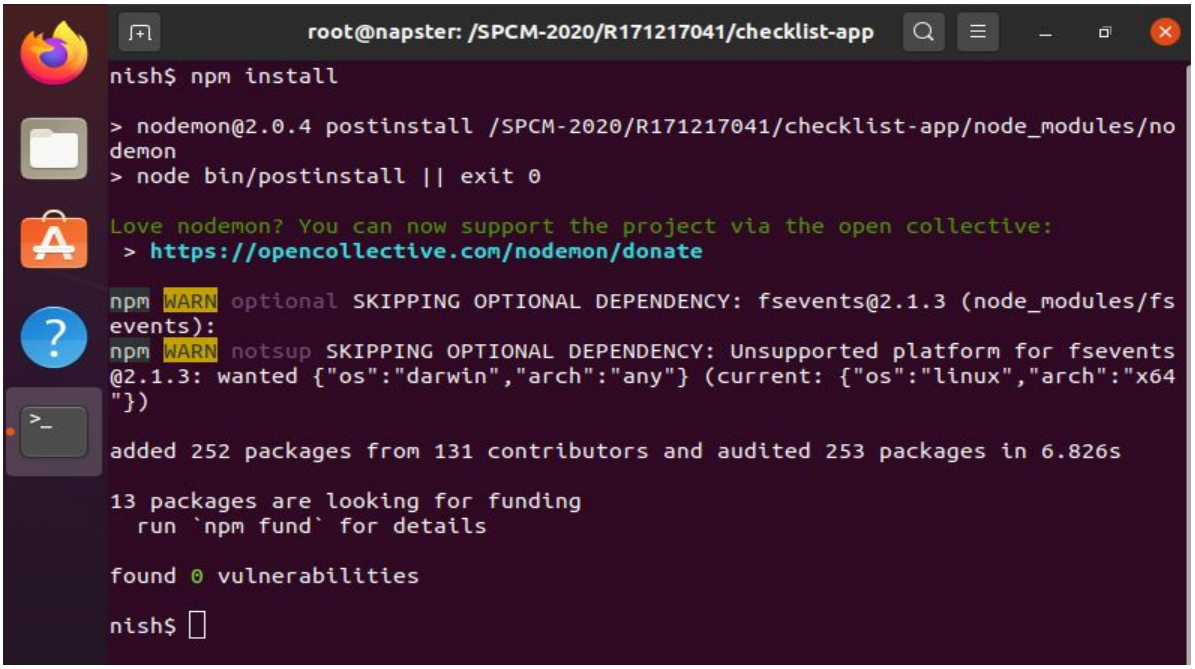
The main files of the project are:

- **package.json** — main server configuration file
- **server.js** — executable script to run the server
- **client/src/App.js** — main JS file for Client application
- **client/src/components/AppNavbar.js**
- **client/src/components/Evangelist.js**

Let's build this application on the local workstation:

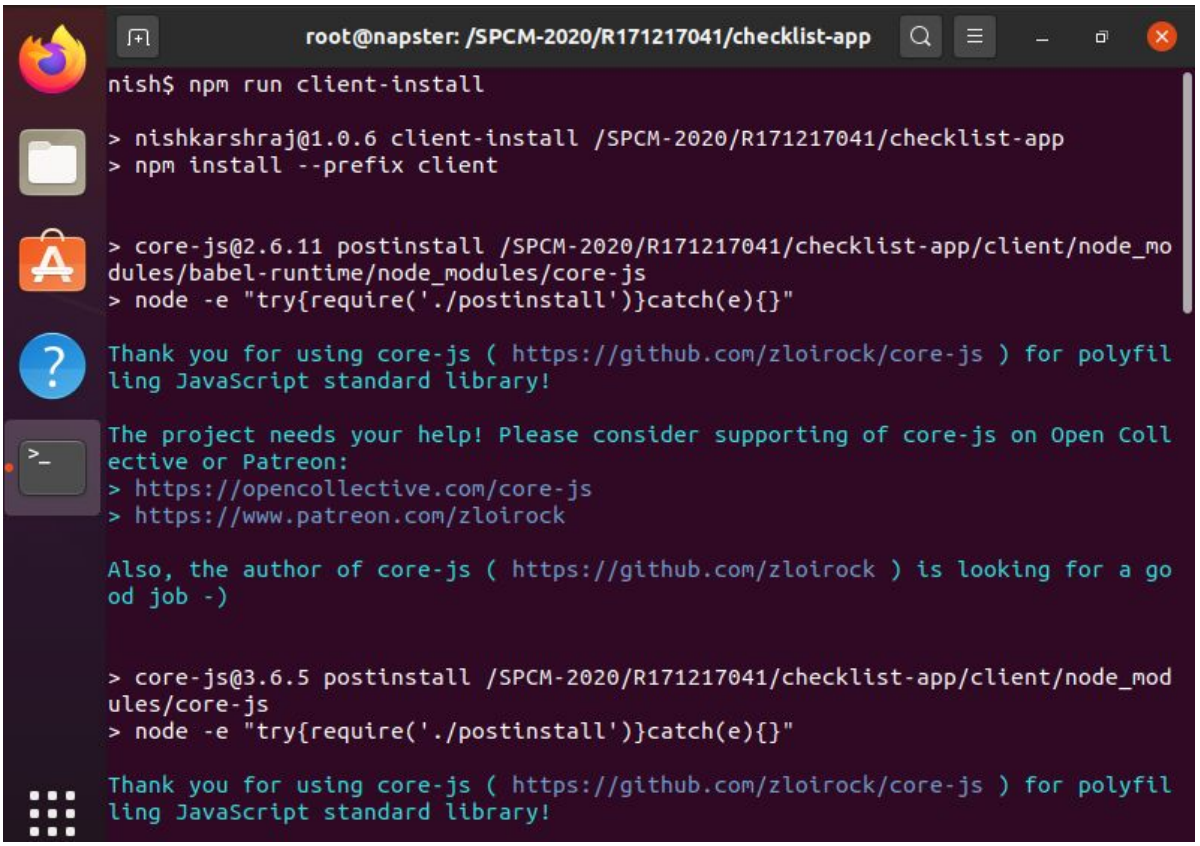
- Install the node modules and dependencies for server application

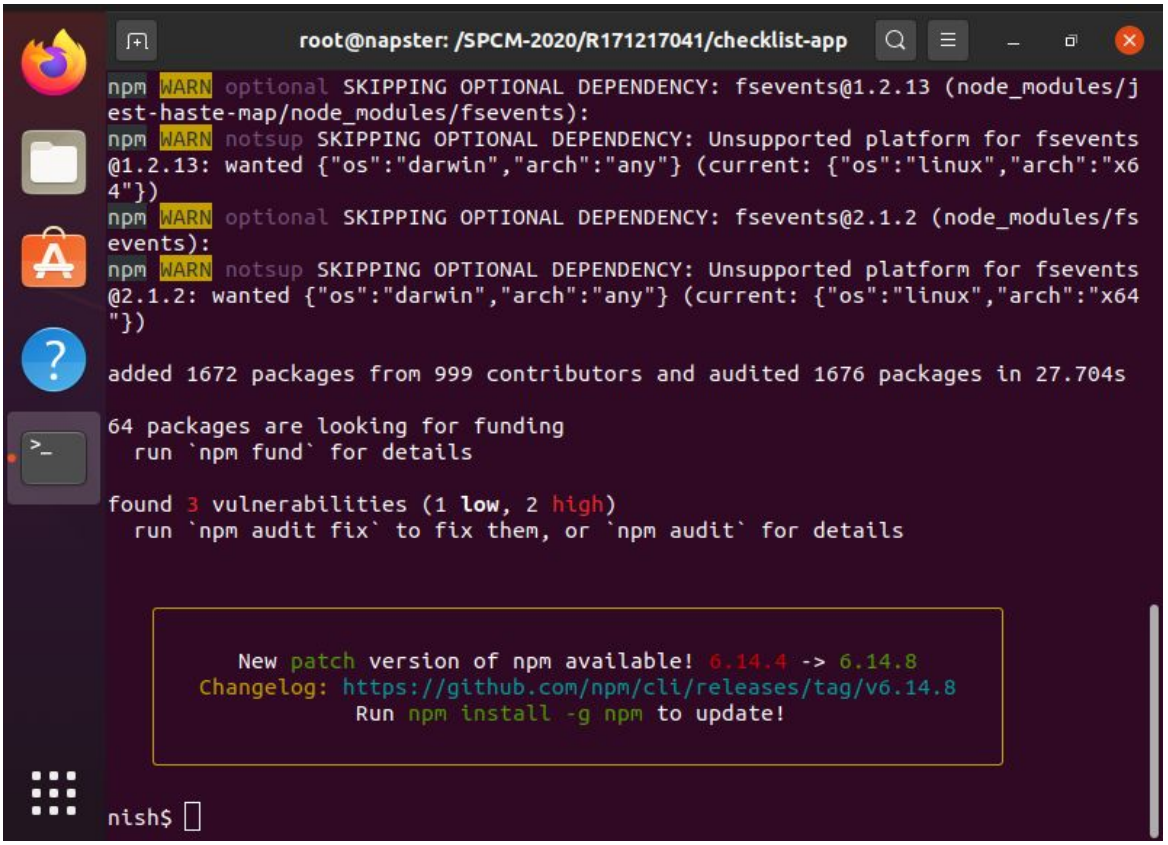
$ npm install



- Install the node modules and dependencies for client application

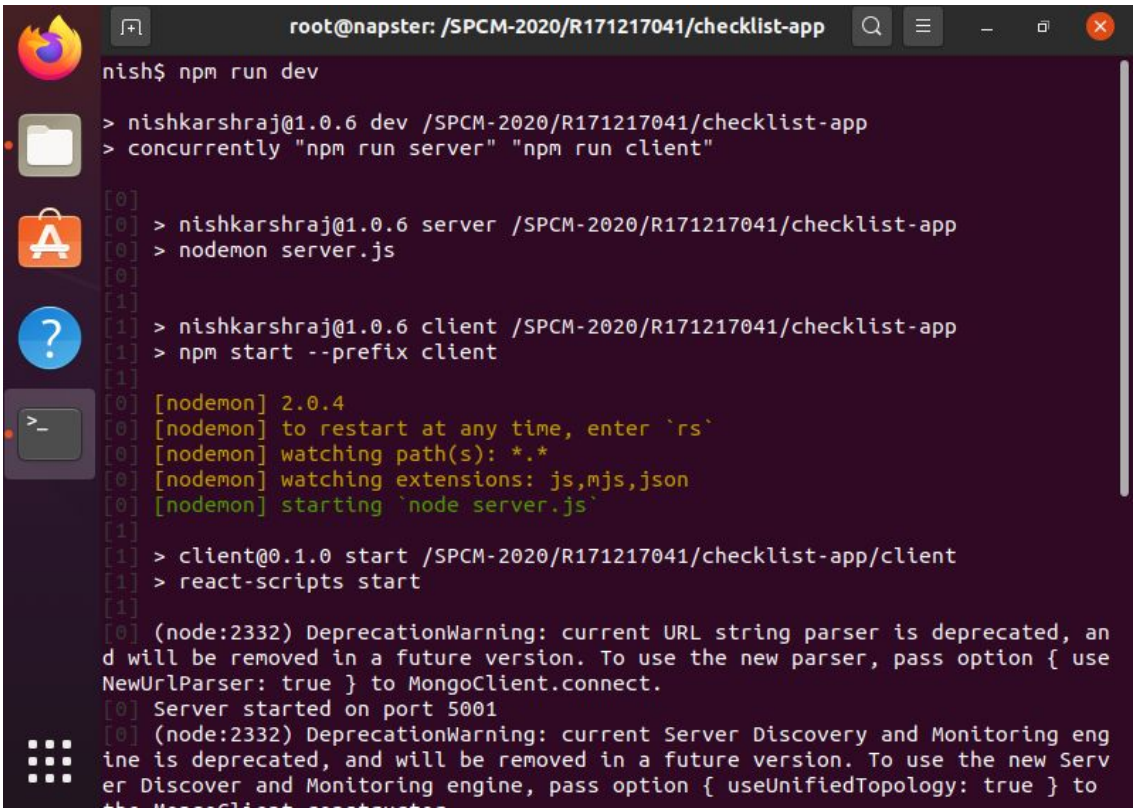
$ npm run client-install

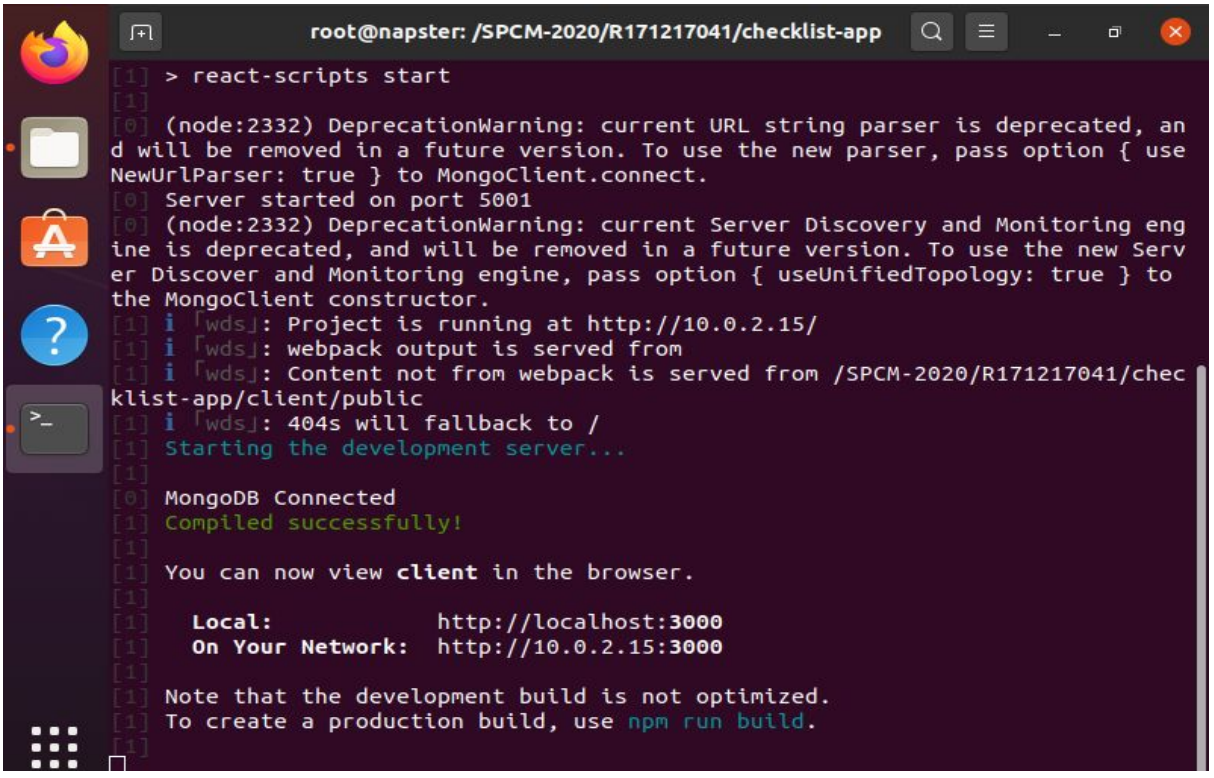




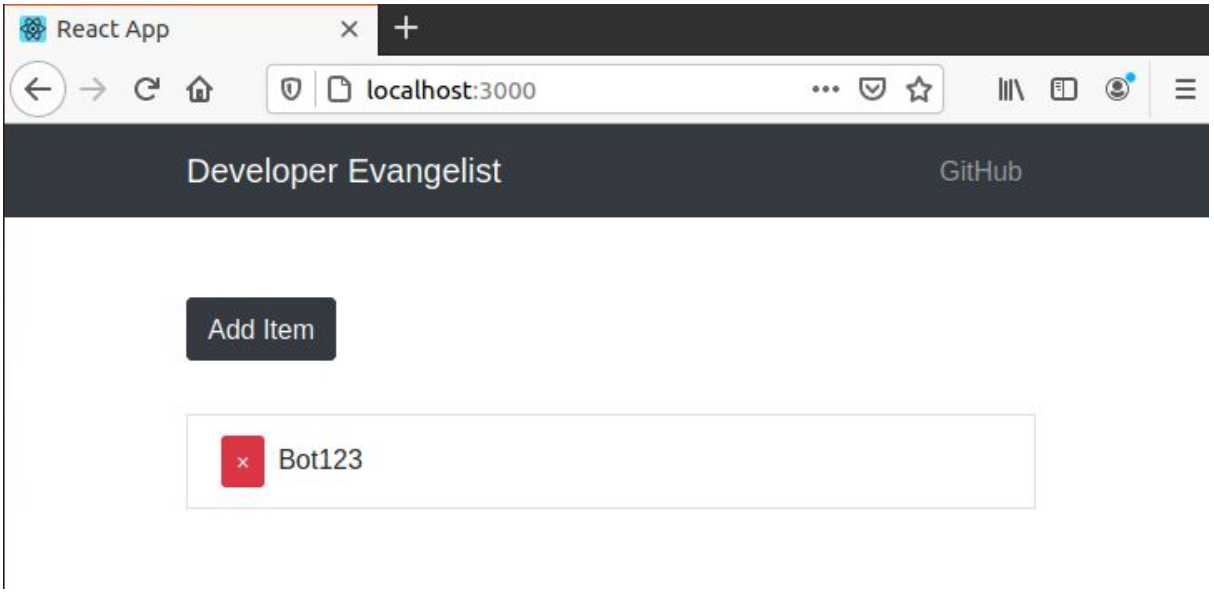
- Run the client and server application concurrently

$ npm run dev

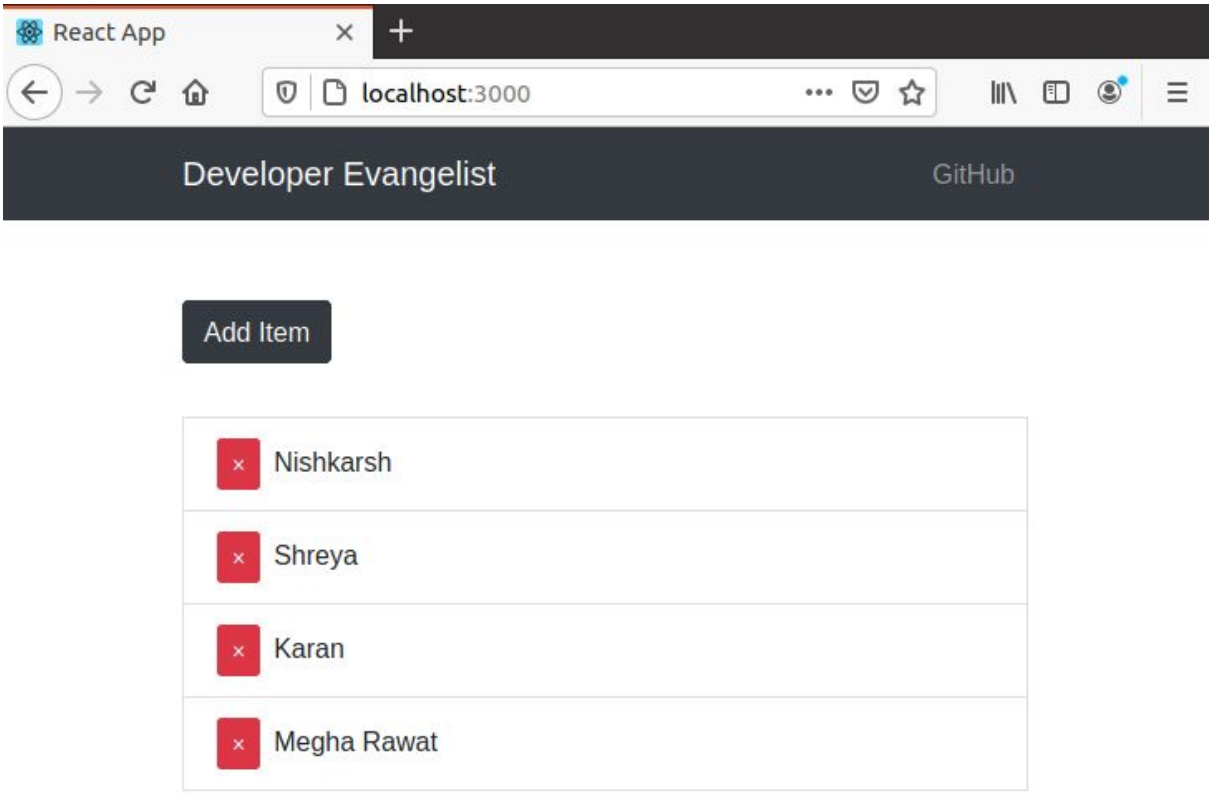




Now, move to the browser and open http://localhost:3000



Add some items and remove the default item.



Step 2) Hosting the web application on GitHub repository

The project is hosted on https://www.github.com/hkshitesh/SPCM-2020/R171217041/

Steps to host a project on GitHub are:

- Initialize local workspace as a git project

$ git init

- Stage the files to be added on GitHub

$ git add [file1] [file2] ...

- Commit changes with an appropriate commit message

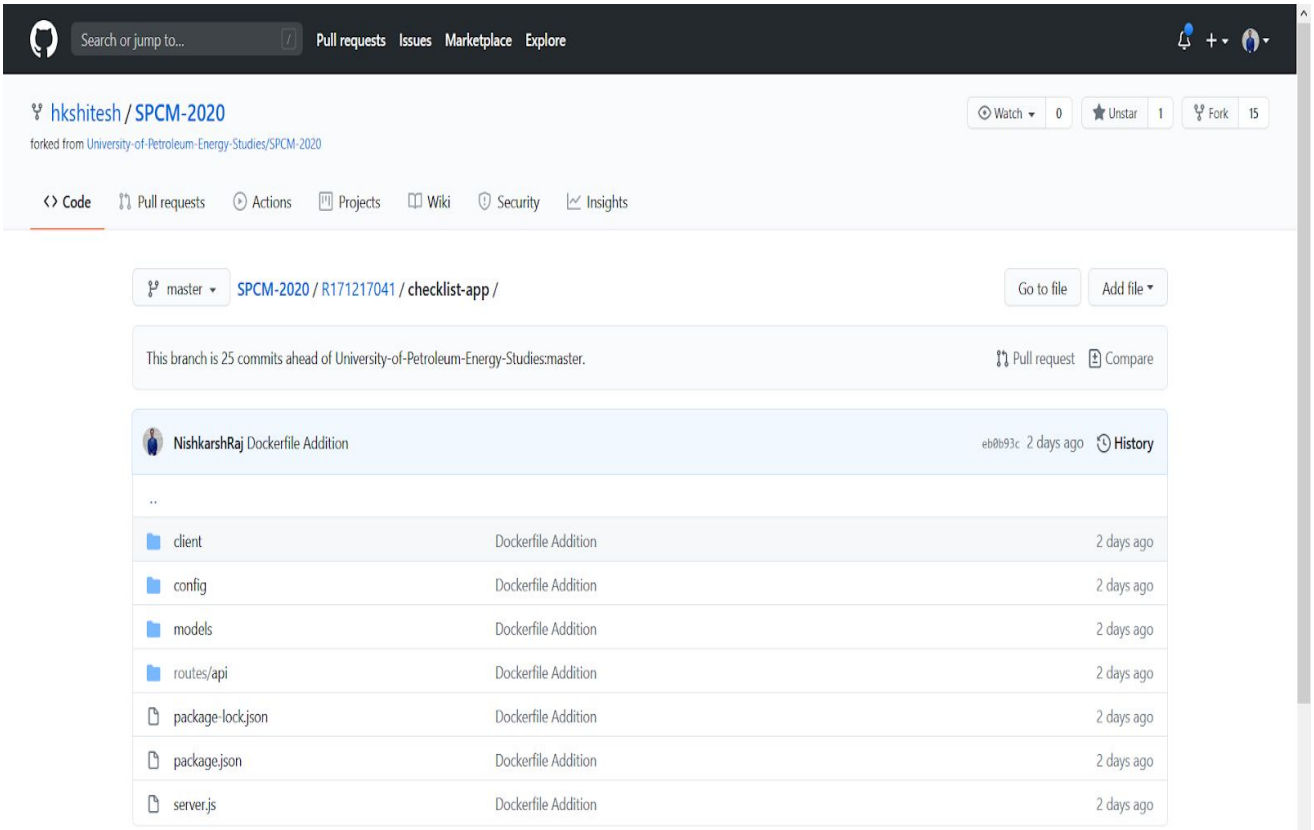
$ git commit -m "<Commit Message>"

- Create an upstream link with your repository on GitHub

$ git remote add origin [Repository URL]

- Push the changes onto GitHub

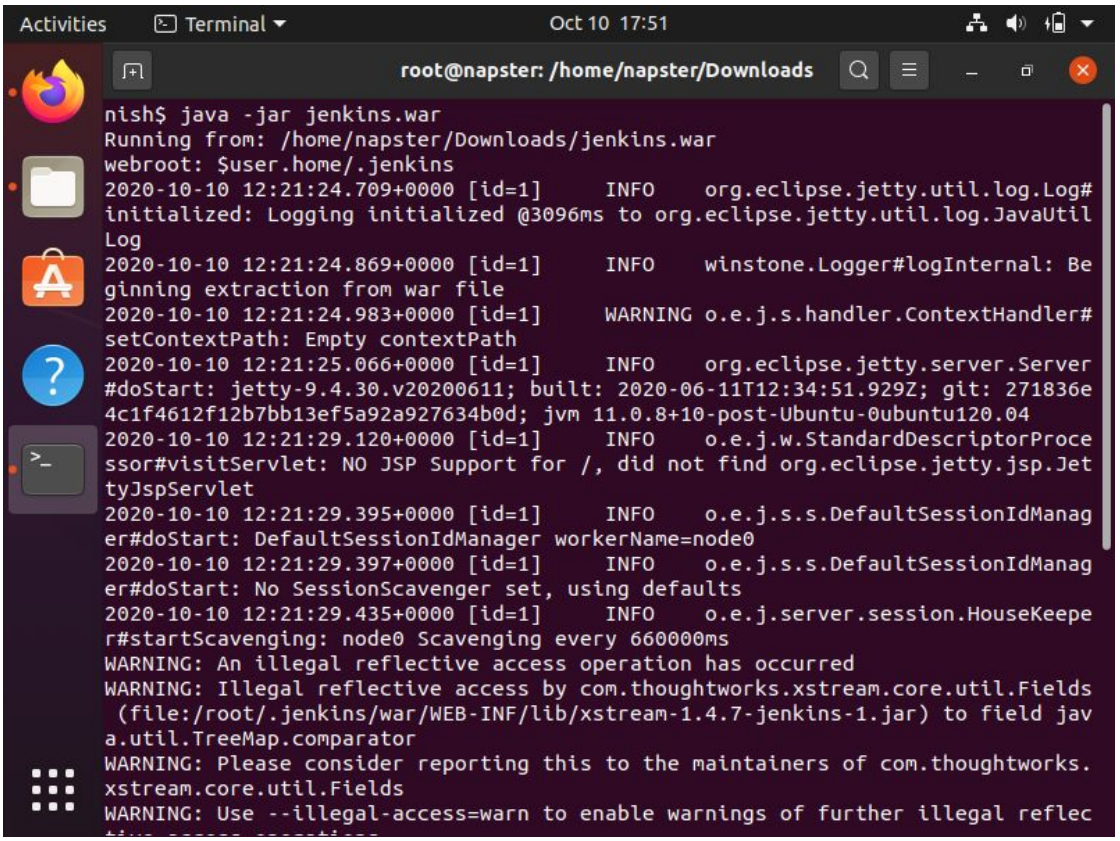
$ git push -u origin master



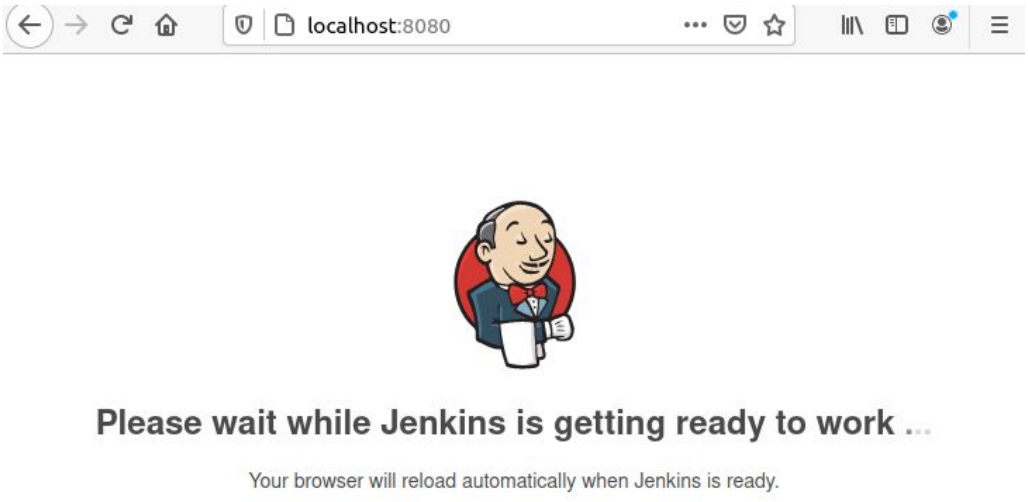
Step 3) Setting up Continuous Integration Build with Jenkins

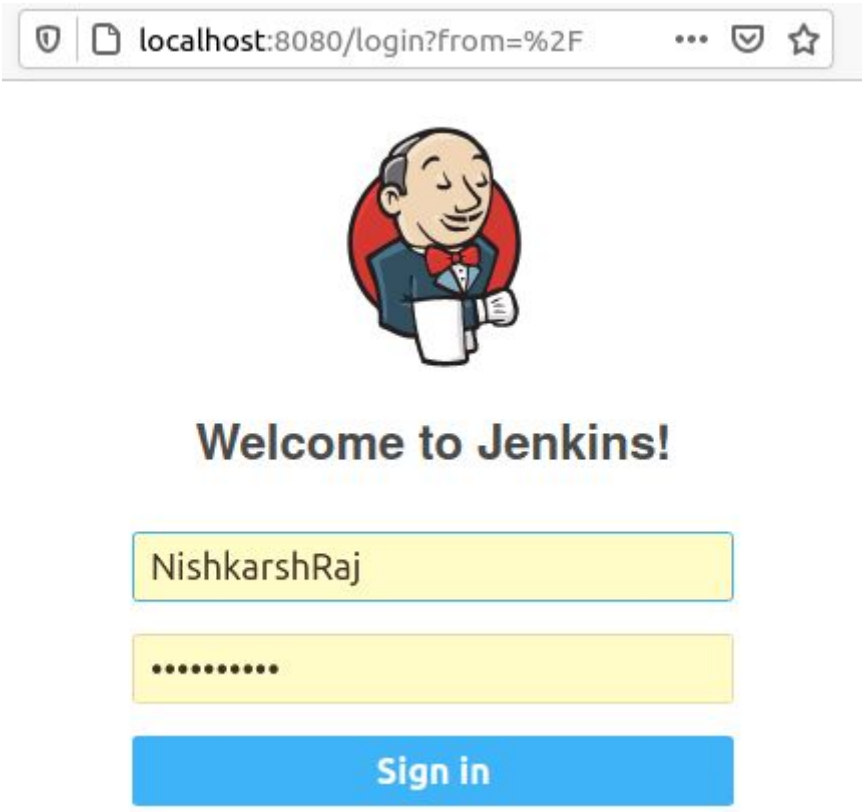
- Start Jenkins server on localhost

$ java -jar jenkins.war

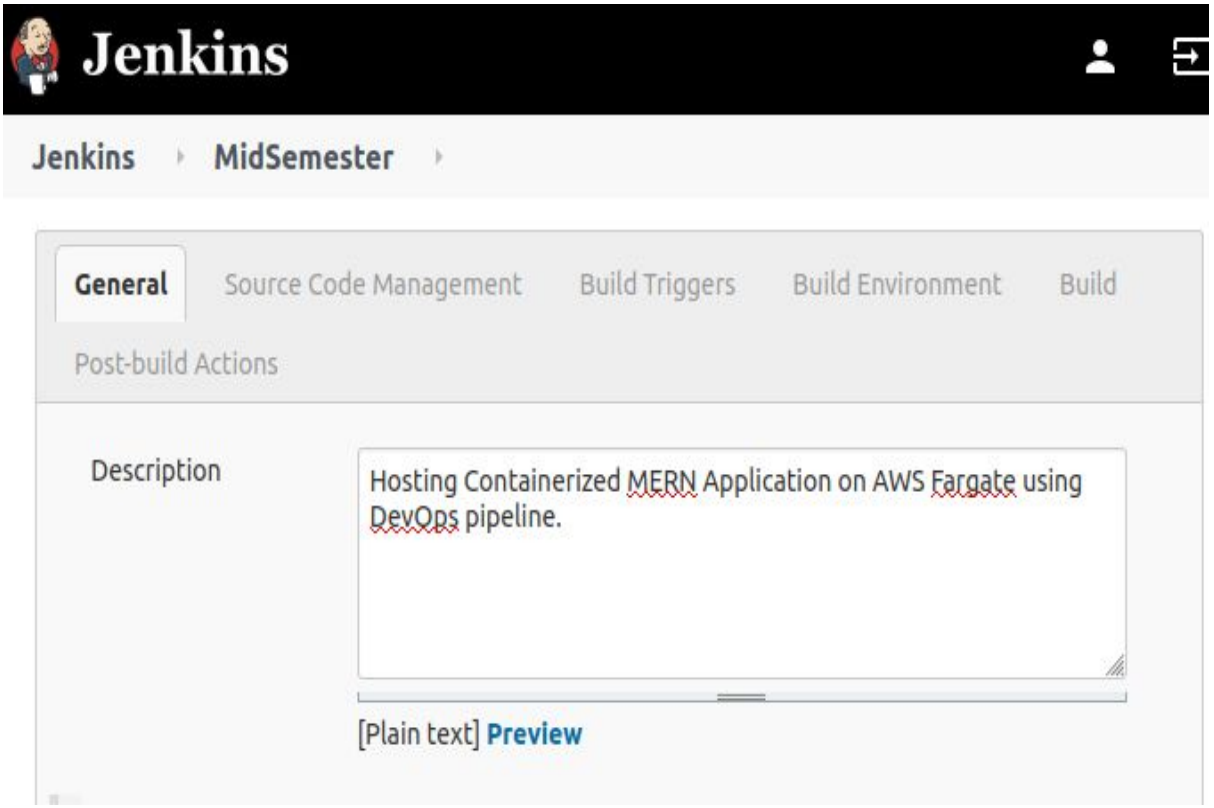


Move to **http://localhost:8080** which is the default hosting of local Jenkins server.

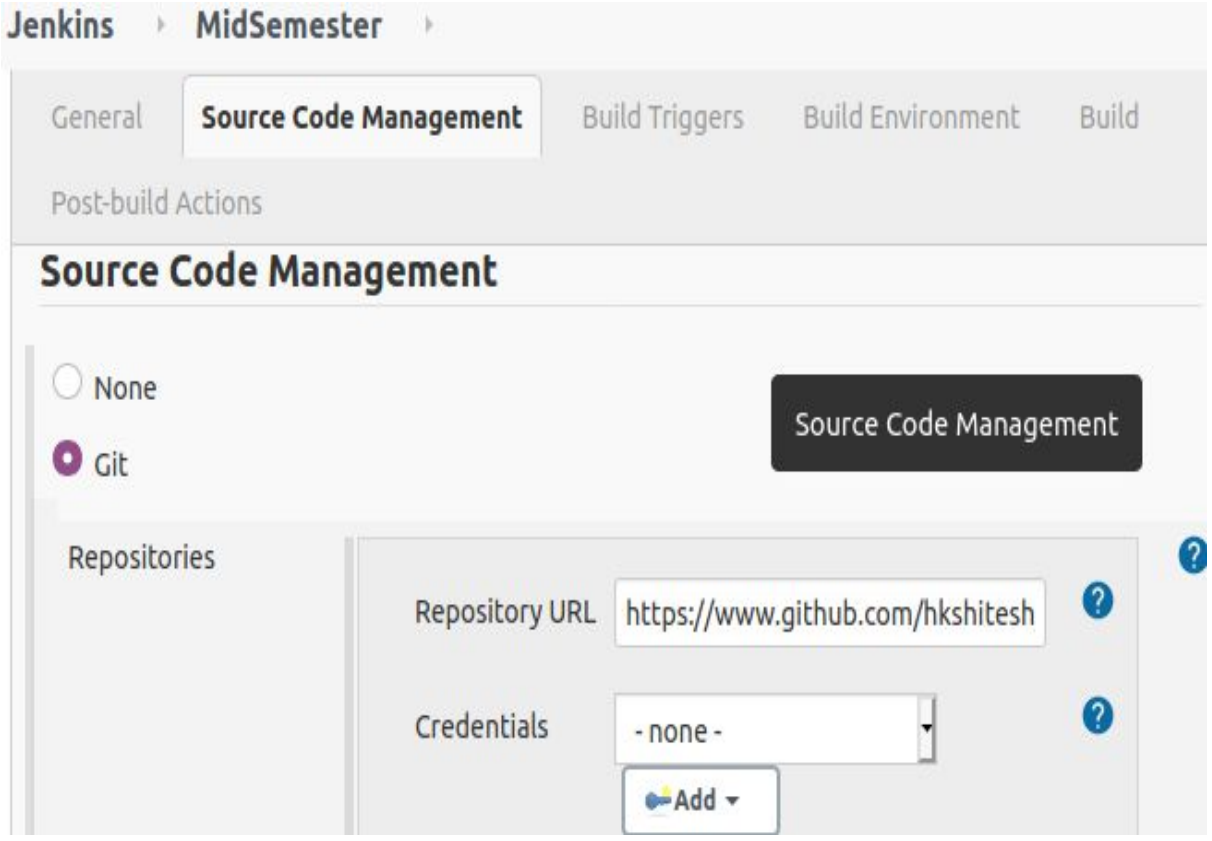




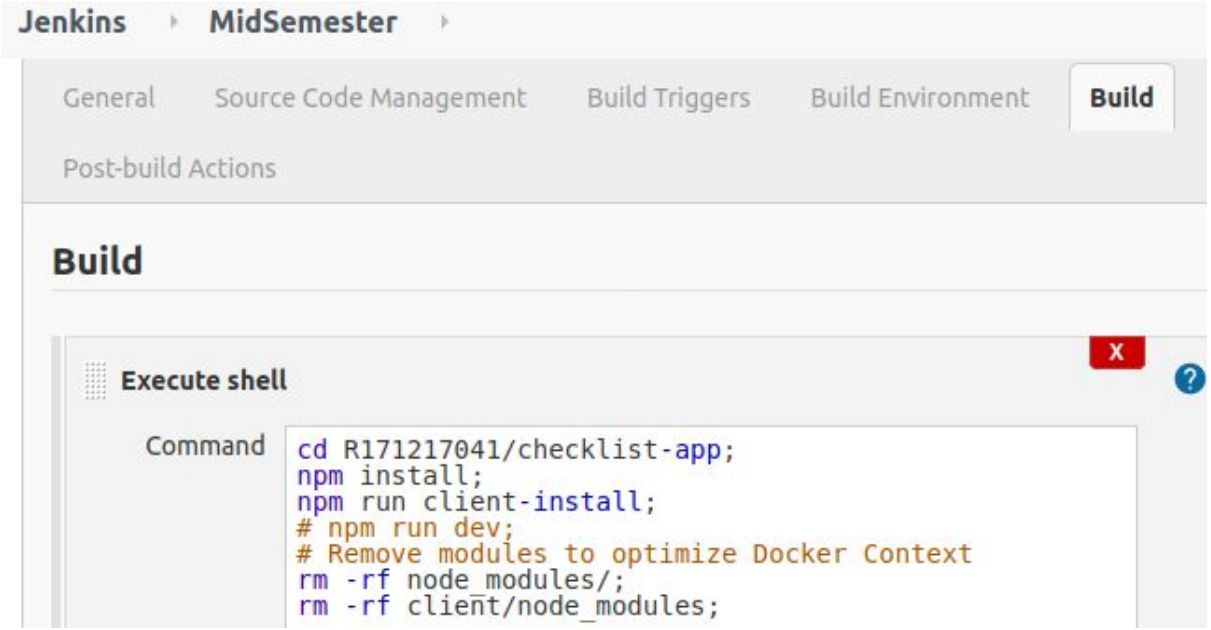
After signing in, create a **New Item -> New Job** with the name and description of your choice.



- Add Git SCM and specify the URL of your GitHub repository

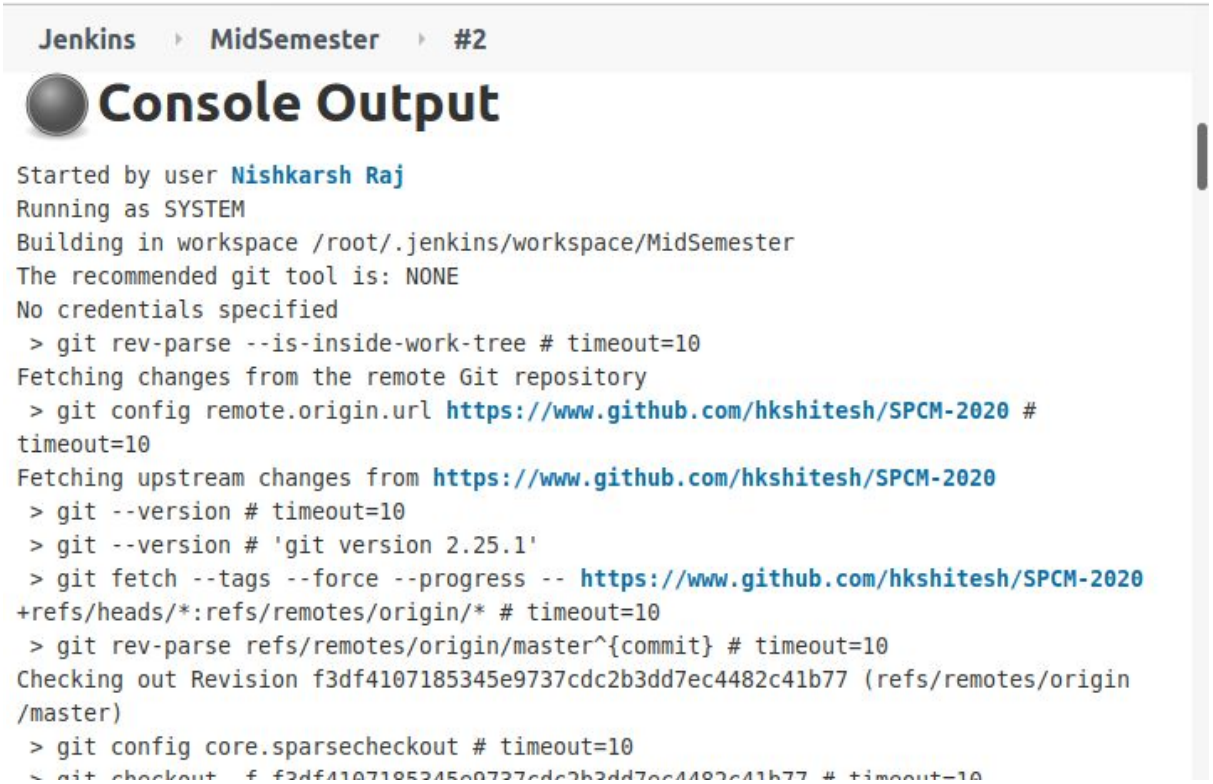


- Create a shell job build action to run the commands of NPM.



The last two lines in the shell commands are used to delete the dependencies to optimize the next step of containerization because Docker sends entire local workspace to the Docker daemon before executing the Dockerfile.

- Save changes. Build the project and see the Console Output



Jenkins › MidSemester › #2

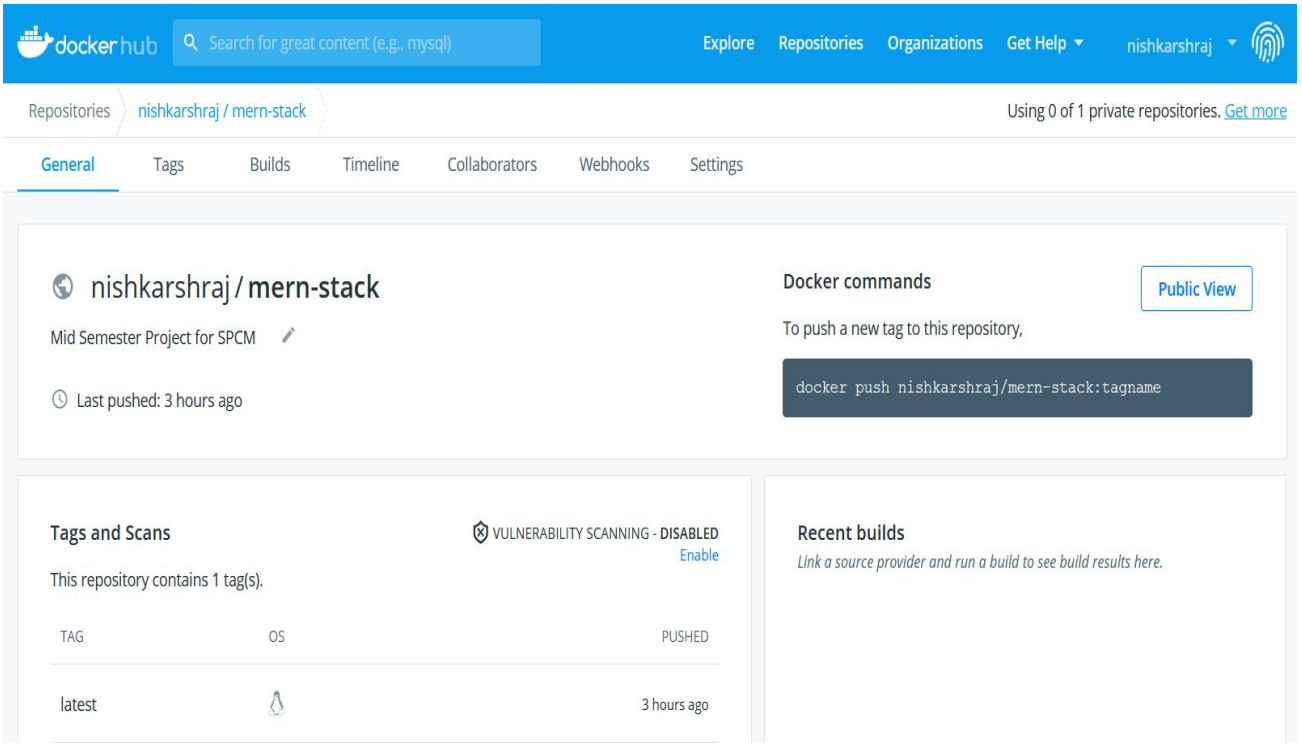
```
+ npm install

> nodemon@2.0.4 postinstall /root/.jenkins/workspace/MidSemester/R171217041
/node_modules/nodemon
> node bin/postinstall || exit 0

internal/modules/cjs/loader.js:638
    throw err;
    ^

Error: Cannot find module '/root/.jenkins/workspace/MidSemester/R171217041
/node_modules/nodemon/bin/postinstall'
    at Function.Module._resolveFilename (internal/modules/cjs/loader.js:636:15)
    at Function.Module._load (internal/modules/cjs/loader.js:562:25)
    at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:623:3)
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3
(node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for
fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current:
```

Jenkins › MidSemester › #2

```
+ npm run client-install

> nishkarshraj@1.0.6 client-install /root/.jenkins/workspace/MidSemester/R171217041
> npm install --prefix client


> core-js@2.6.11 postinstall /root/.jenkins/workspace/MidSemester/R171217041/client
/node_modules/babel-runtime/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"


> core-js@3.6.5 postinstall /root/.jenkins/workspace/MidSemester/R171217041/client
/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"


> core-js-pure@3.6.5 postinstall /root/.jenkins/workspace/MidSemester/R171217041
/client/node_modules/core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"
```

Step 4) Containerizing the MERN Application

Before containerizing the application, create a repository for the Docker image on DockerHub.

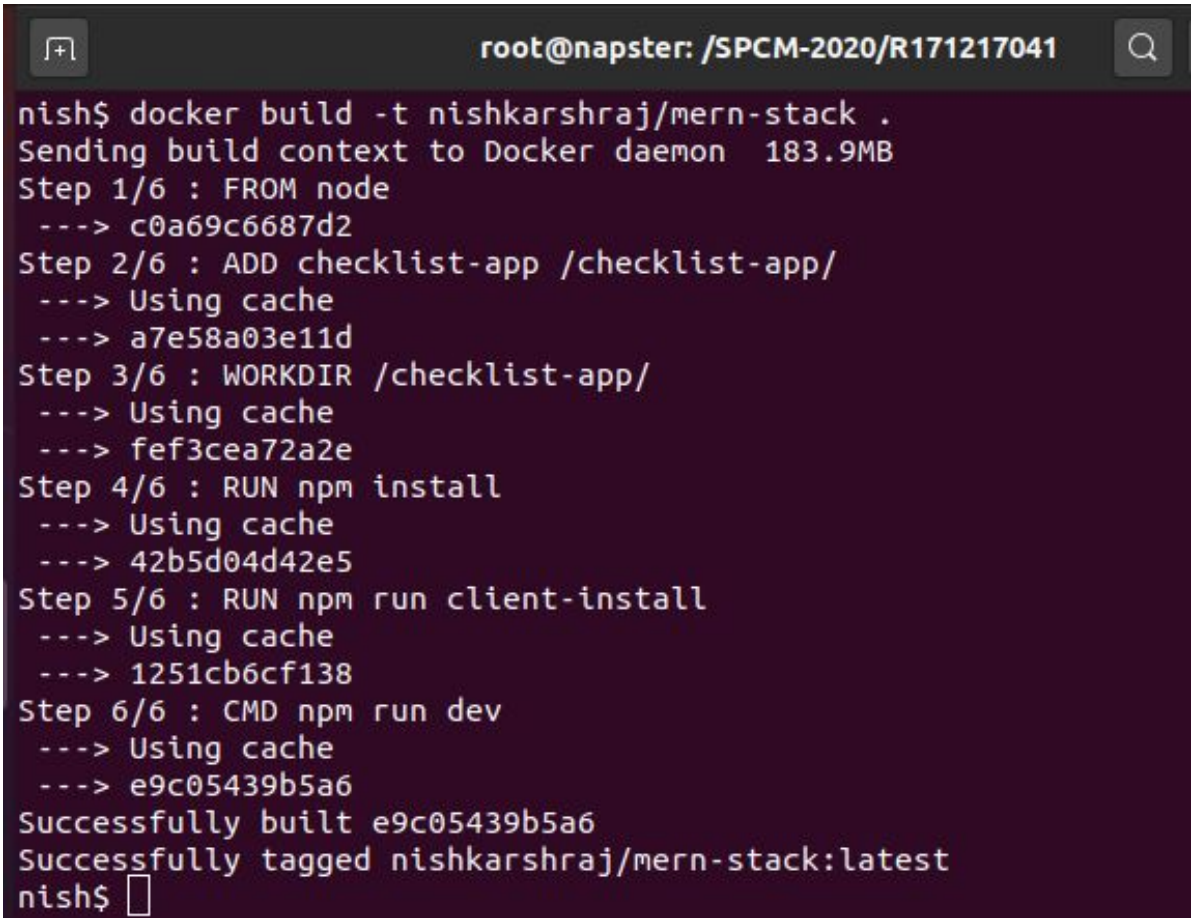


- Create a Dockerfile with Node as the base image

RUN vs CMD Command: RUN command specifies tasks to be executed when the docker image is created while the CMD command specifies the tasks to be executed when the docker container is launched.

- Create the Docker Image

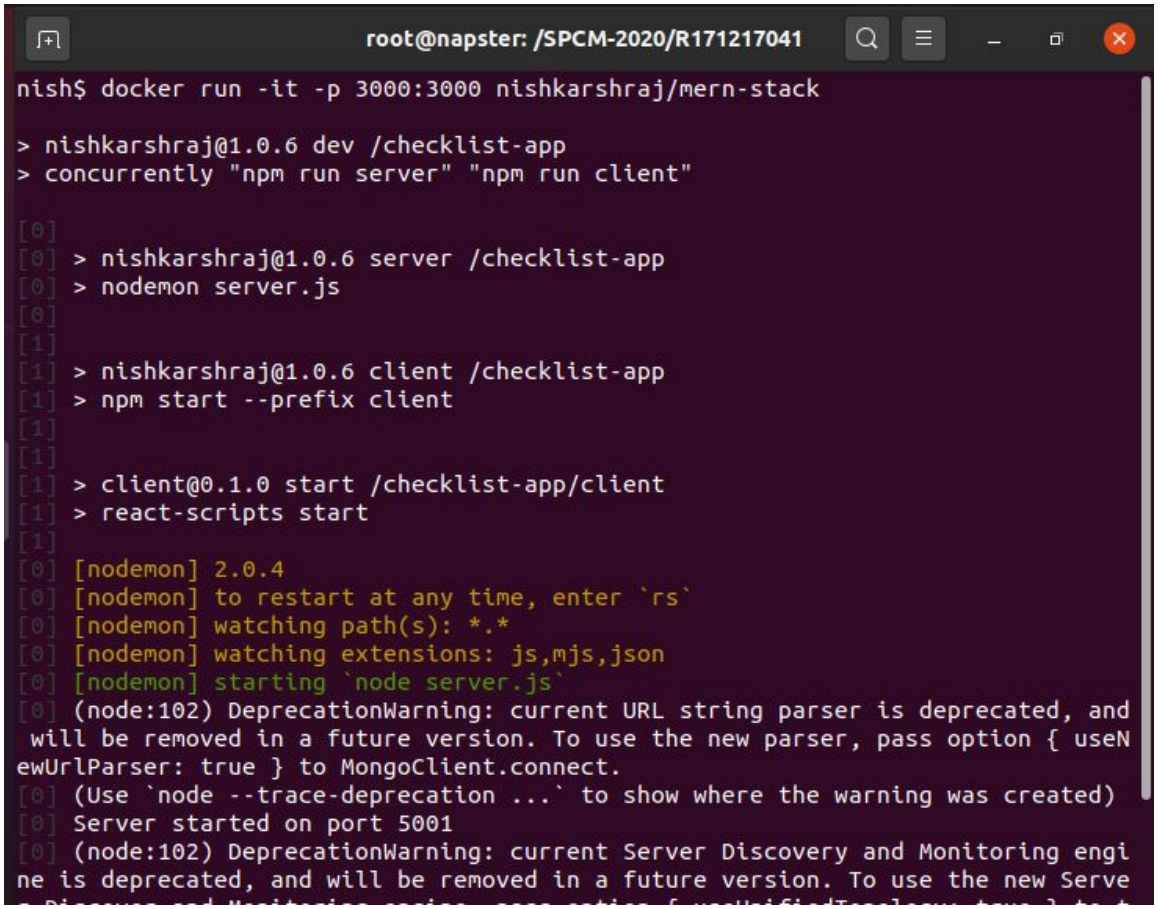
$ docker build -t nishkarshraj/mern-stack .

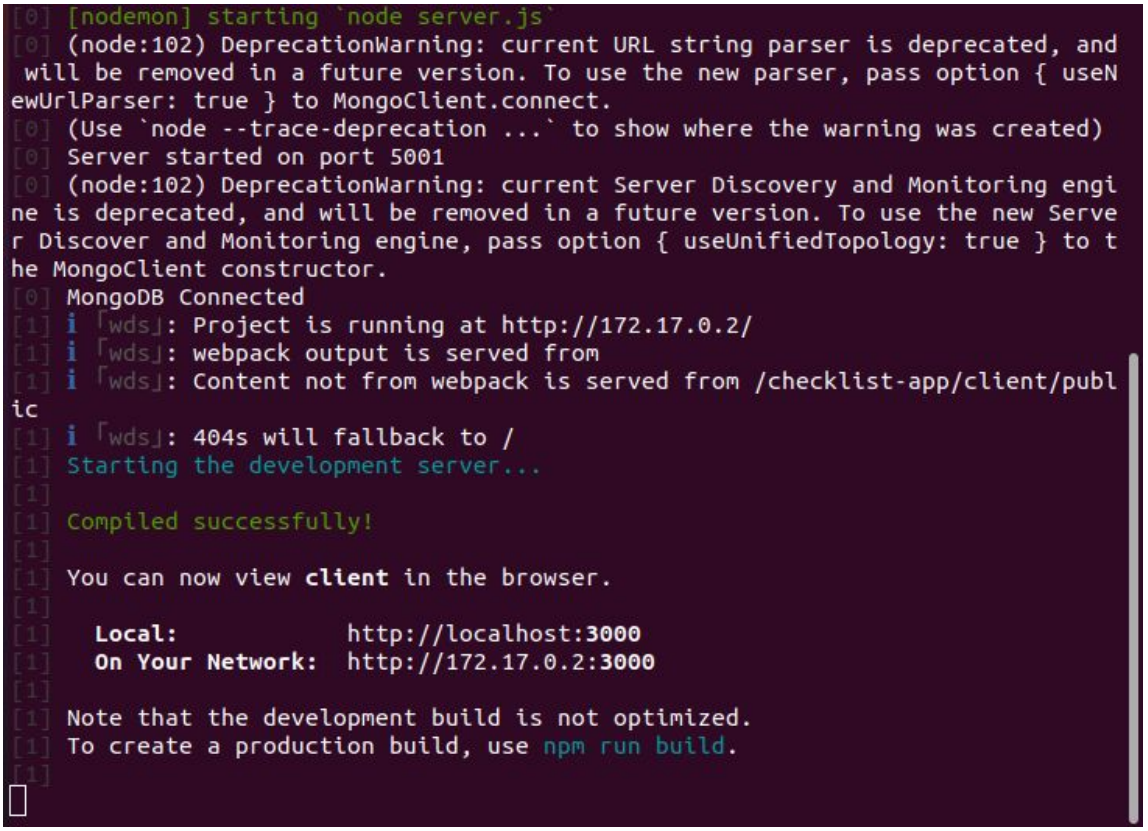


- Launch the Docker Container to test the application locally

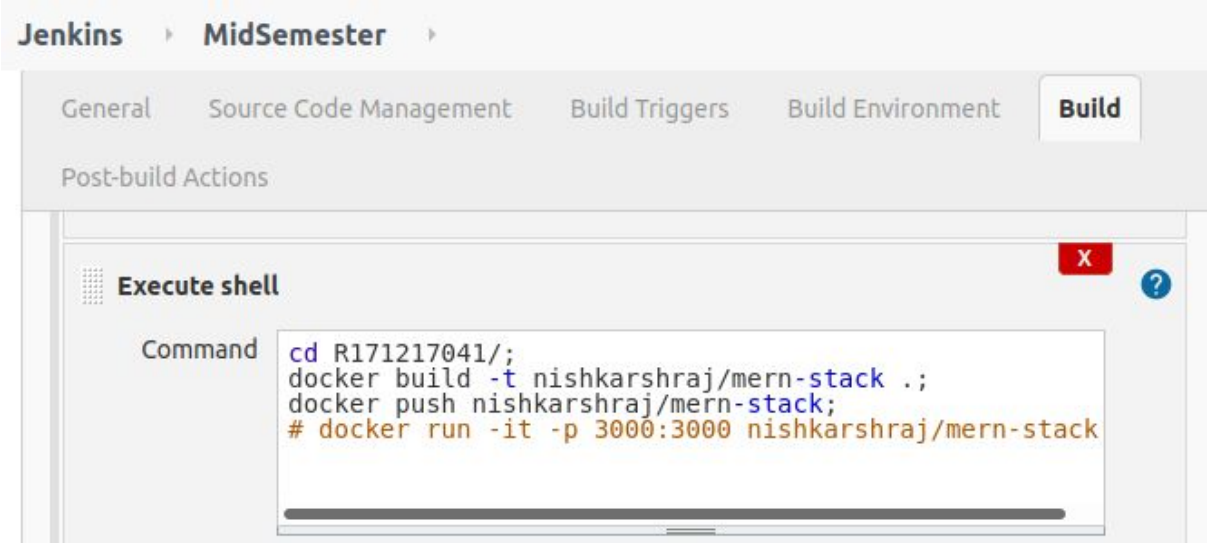
$ docker run -it -p 3000:3000 nishkarshraj/mern-stack

The application runs in interactive mode with port sharing of 3000 port.

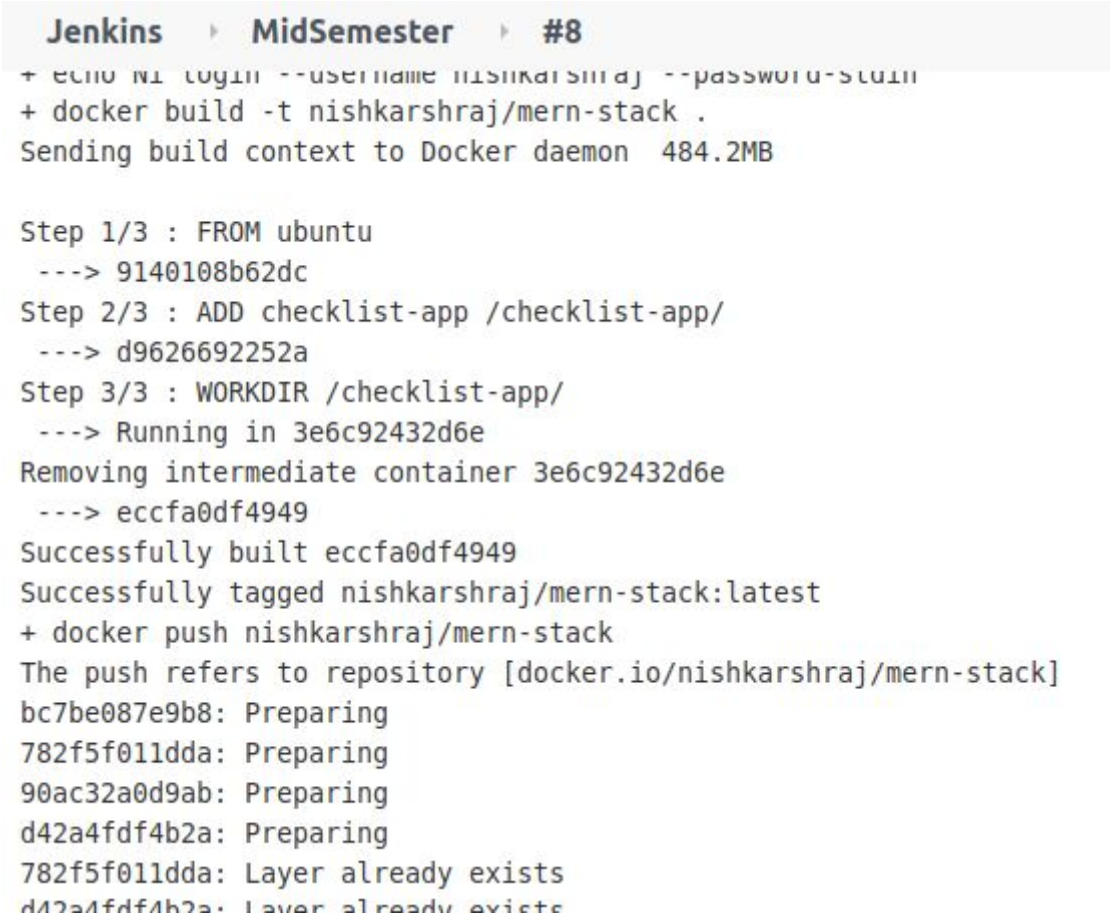

root@napster: /SPCM-2020/R171217041
nish$ docker run -it -p 3000:3000 nishkarshraj/mern-stack
> nishkarshraj@1.0.6 dev /checklist-app
> concurrently "npm run server" "npm run client"
[0]
[0] > nishkarshraj@1.0.6 server /checklist-app
[0] > nodemon server.js
[0]
[1]
[1] > nishkarshraj@1.0.6 client /checklist-app
[1] > npm start --prefix client
[1]
[1]
[1] > client@0.1.0 start /checklist-app/client
[1] > react-scripts start
[1]
[0] [nodemon] 2.0.4
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching path(s): *.*
[0] [nodemon] watching extensions: js,mjs,json
[0] [nodemon] starting `node server.js`
[0] (node:102) DeprecationWarning: current URL string parser is deprecated, and
will be removed in a future version. To use the new parser, pass option { useN
ewUrlParser: true } to MongoClient.connect.
[0] (Use `node --trace-deprecation ...` to show where the warning was created)
[0] Server started on port 5001
[0] (node:102) DeprecationWarning: current Server Discovery and Monitoring engi
ne is deprecated, and will be removed in a future version. To use the new Serve


[0] [nodemon] starting `node server.js`
[0] (node:102) DeprecationWarning: current URL string parser is deprecated, and
will be removed in a future version. To use the new parser, pass option { useN
ewUrlParser: true } to MongoClient.connect.
[0] (Use `node --trace-deprecation ...` to show where the warning was created)
[0] Server started on port 5001
[0] (node:102) DeprecationWarning: current Server Discovery and Monitoring engi
ne is deprecated, and will be removed in a future version. To use the new Serve
r Discover and Monitoring engine, pass option { useUnifiedTopology: true } to t
he MongoClient constructor.
[0] MongoDB Connected
[1] i [wds]: Project is running at http://172.17.0.2/
[1] i [wds]: webpack output is served from
[1] i [wds]: Content not from webpack is served from /checklist-app/client/publ
ic
[1] i [wds]: 404s will fallback to /
[1] Starting the development server...
[1]
[1] Compiled successfully!
[1]
[1] You can now view client in the browser.
[1]
[1] Local: http://localhost:3000
[1] On Your Network: http://172.17.0.2:3000
[1]
[1] Note that the development build is not optimized.
[1] To create a production build, use npm run build.
[1]

- Create Docker Build and Push task on Jenkins and push the Docker image on DockerHub



Let's see the console log for this change.



```
Jenkins ▸ MidSemester ▸ #8
Step 3/3 : WORKDIR /checklist-app/
 ---> Running in 3e6c92432d6e
Removing intermediate container 3e6c92432d6e
 ---> eccfa0df4949
Successfully built eccfa0df4949
Successfully tagged nishkarshraj/mern-stack:latest
+ docker push nishkarshraj/mern-stack
The push refers to repository [docker.io/nishkarshraj/mern-stack]
bc7be087e9b8: Preparing
782f5f011dda: Preparing
90ac32a0d9ab: Preparing
d42a4fdf4b2a: Preparing
782f5f011dda: Layer already exists
d42a4fdf4b2a: Layer already exists
90ac32a0d9ab: Layer already exists
bc7be087e9b8: Pushed
latest: digest:
sha256:3e75dd06908d367513bacd75f4292889567611fd45d8fd498213d1fd268fad50 size: 1155
Finished: SUCCESS
```

Step 5) Setting up infrastructure automation using Terraform and AWS Cloud

- Install AWS-CLI package which is used to store AWS IAM user credentials securely and de-couples them from the terraform scripts.

$ apt install -y awscli

- Create the credentials file

$ aws configure

Terraform is used to create infrastructure automation on AWS Cloud by creating a Fargate cluster and running our Docker Image remotely.

Let's look at the terraform script files used in this project:

- **provider.tf**
- **variables.tf**
- **security.tf**
- **ecs.tf**

Let's check the Terraform configurations locally.

- Initialize terraform project and install dependencies.

$ terraform init

```
nish$ terraform init

Initializing the backend...

Initializing provider plugins...
- Using previously-installed hashicorp/template v2.2.0
- Using previously-installed hashicorp/aws v3.10.0

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, we recommend adding version constraints in a required_providers block
in your configuration, with the constraint strings suggested below.

* hashicorp/aws: version = "~> 3.10.0"
* hashicorp/template: version = "~> 2.2.0"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
nish$
```

- Create plan and see the changes that would occur on cloud after apply event

$ terraform plan

```
nish$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.template_file.myapp: Refreshing state...
data.aws_iam_policy_document.ecs_task_execution_role: Refreshing state...
data.aws_availability_zones.available: Refreshing state...

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_alb.main will be created
  + resource "aws_alb" "main" {
      + arn                        = (known after apply)
      + arn_suffix                 = (known after apply)
      + dns_name                   = (known after apply)
      + drop_invalid_header_fields = false
      + enable_deletion_protection = false
      + enable_http2               = true
      + id                         = (known after apply)
      + idle_timeout               = 60
      + internal                   = (known after apply)
      + ip_address_type            = (known after apply)
```

```
      + cidr_block                     = "172.17.0.0/16"
      + default_network_acl_id         = (known after apply)
      + default_route_table_id         = (known after apply)
      + default_security_group_id      = (known after apply)
      + dhcp_options_id                = (known after apply)
      + enable_classiclink             = (known after apply)
      + enable_classiclink_dns_support = (known after apply)
      + enable_dns_hostnames           = (known after apply)
      + enable_dns_support             = true
      + id                             = (known after apply)
      + instance_tenancy               = "default"
      + ipv6_association_id            = (known after apply)
      + ipv6_cidr_block                = (known after apply)
      + main_route_table_id            = (known after apply)
      + owner_id                       = (known after apply)
    }

Plan: 32 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + alb_hostname = (known after apply)

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

- Apply the changes and host project on the cloud

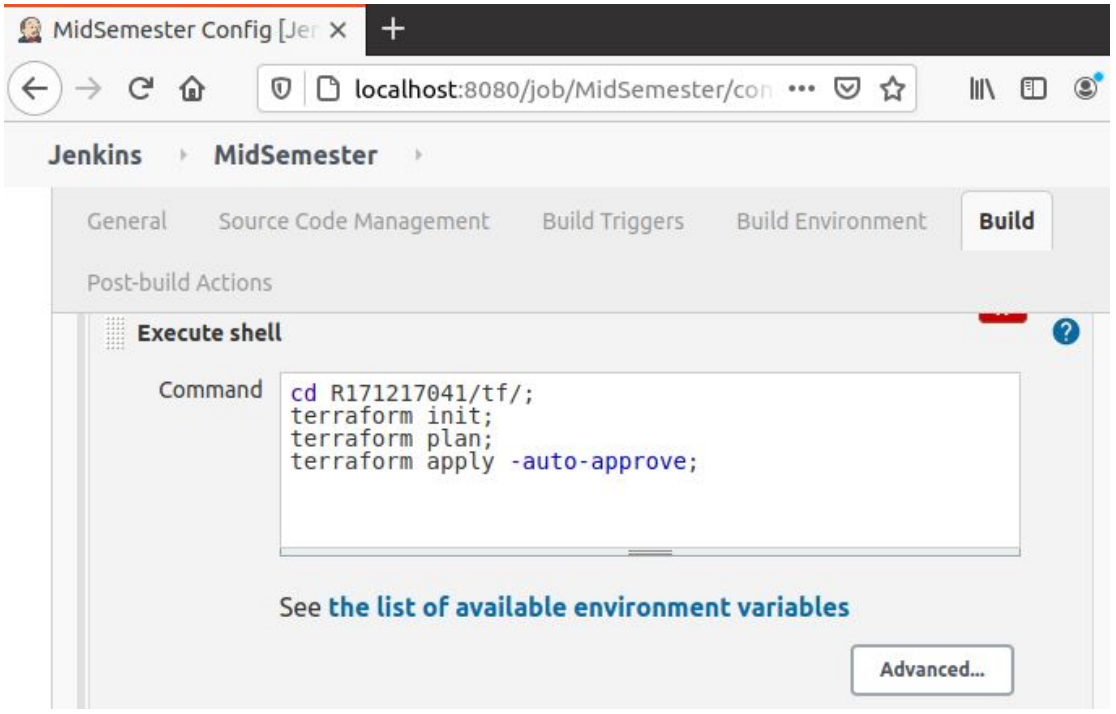
$ terraform apply -auto-approve

```
nish$ terraform apply -auto-approve
data.template_file.myapp: Refreshing state...
data.aws_availability_zones.available: Refreshing state...
data.aws_iam_policy_document.ecs_task_execution_role: Refreshing state...
aws_iam_role.ecs_task_execution_role: Creating...
aws_cloudwatch_log_group.myapp_log_group: Creating...
aws_ecs_cluster.main: Creating...
aws_vpc.main: Creating...
aws_cloudwatch_log_group.myapp_log_group: Creation complete after 0s [id=/ecs/m
yapp]
aws_cloudwatch_log_stream.myapp_log_stream: Creating...
aws_cloudwatch_log_stream.myapp_log_stream: Creation complete after 0s [id=my-l
og-stream]
aws_vpc.main: Creation complete after 2s [id=vpc-096431bef00cd5324]
aws_subnet.public[1]: Creating...
aws_subnet.public[0]: Creating...
aws_internet_gateway.gw: Creating...
aws_subnet.private[1]: Creating...
aws_alb_target_group.app: Creating...
aws_security_group.lb: Creating...
aws_subnet.private[0]: Creating...
aws_iam_role.ecs_task_execution_role: Creation complete after 2s [id=myEcsTaskE
xecutionRole]
aws_iam_role_policy_attachment.ecs_task_execution_role: Creating...
aws_ecs_task_definition.app: Creating...
aws_ecs_task_definition.app: Creation complete after 1s [id=myapp-task]
aws_subnet.private[0]: Creation complete after 1s [id=subnet-0851aa0b40c013497]
aws_subnet.private[1]: Creation complete after 1s [id=subnet-037d686d63550ca48]
aws_internet_gateway.gw: Creation complete after 1s [id=igw-0be98d875adc51e28]
```

```
-south-1:579354883343:loadbalancer/app/myapp-load-balancer/2cd3b45a3ae2909a]
aws_alb_listener.front_end: Creating...
aws_alb_listener.front_end: Creation complete after 0s [id=arn:aws:elasticloadb
alancing:ap-south-1:579354883343:listener/app/myapp-load-balancer/2cd3b45a3ae29
09a/ed7079b524d30094]
aws_ecs_service.main: Creating...
aws_ecs_service.main: Creation complete after 2s [id=arn:aws:ecs:ap-south-1:579
354883343:service/myapp-service]
aws_appautoscaling_target.target: Creating...
aws_appautoscaling_target.target: Creation complete after 1s [id=service/myapp-
cluster/myapp-service]
aws_appautoscaling_policy.down: Creating...
aws_appautoscaling_policy.up: Creating...
aws_appautoscaling_policy.down: Creation complete after 0s [id=myapp_scale_down
]
aws_cloudwatch_metric_alarm.service_cpu_low: Creating...
aws_appautoscaling_policy.up: Creation complete after 0s [id=myapp_scale_up]
aws_cloudwatch_metric_alarm.service_cpu_high: Creating...
aws_cloudwatch_metric_alarm.service_cpu_high: Creation complete after 1s [id=my
app_cpu_utilization_high]
aws_cloudwatch_metric_alarm.service_cpu_low: Creation complete after 1s [id=mya
pp_cpu_utilization_low]

Apply complete! Resources: 32 added, 0 changed, 0 destroyed.

Outputs:

alb_hostname = myapp-load-balancer-1488972820.ap-south-1.elb.amazonaws.com
nish$
```

The highlighted link is where our containerized project is hosted on the Fargate cluster. Click on it to view the web application online.

- Creating Jenkins automation for Terraform configurations



- Check the Console Output

Jenkins › MidSemester › #10

```
+ terraform init

[0m[1mInitializing the backend...[0m

[0m[1mInitializing provider plugins...[0m
- Finding latest version of hashicorp/aws...
- Finding latest version of hashicorp/template...
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)
- Installing hashicorp/aws v3.10.0...
- Installed hashicorp/aws v3.10.0 (signed by HashiCorp)

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, we recommend adding version constraints in a required_providers block
in your configuration, with the constraint strings suggested below.

* hashicorp/aws: version = "~> 3.10.0"
* hashicorp/template: version = "~> 2.2.0"
```

Jenkins › MidSemester › #10

```
+ terraform plan
[0m[1mRefreshing Terraform state in-memory prior to plan...[0m
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.
[0m
[0m[1mdata.template_file.myapp: Refreshing state...[0m
[0m[1mdata.aws_iam_policy_document.ecs_task_execution_role: Refreshing state...[0m
[0m[1mdata.aws_availability_zones.available: Refreshing state...[0m

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  [32m+[0m create
[0m
Terraform will perform the following actions:

[1m  # aws_alb.main[0m will be created[0m[0m
[0m  [32m+[0m[0m resource "aws_alb" "main" {
      [32m+[0m [0m[1m[0marn[0m[0m                        = (known after apply)
      [32m+[0m [0m[1m[0marn_suffix[0m[0m                 = (known after apply)
```

```
+ terraform apply -auto-approve
[0m[1mdata.template_file.myapp: Refreshing state...[0m
[0m[1mdata.aws_availability_zones.available: Refreshing state...[0m
[0m[1mdata.aws_iam_policy_document.ecs_task_execution_role: Refreshing state...[0m
[0m[1maws_vpc.main: Creating...[0m[0m
[0m[1maws_ecs_cluster.main: Creating...[0m[0m
[0m[1maws_cloudwatch_log_group.myapp_log_group: Creating...[0m[0m
[0m[1maws_iam_role.ecs_task_execution_role: Creating...[0m[0m
[0m[1maws_cloudwatch_log_group.myapp_log_group: Creation complete after 1s [id=/ecs
/myapp][0m[0m
[0m[1maws_cloudwatch_log_stream.myapp_log_stream: Creating...[0m[0m
[0m[1maws_cloudwatch_log_stream.myapp_log_stream: Creation complete after 0s [id=my-
log-stream][0m[0m
[0m[1maws_iam_role.ecs_task_execution_role: Creation complete after 2s
[id=myEcsTaskExecutionRole][0m[0m
[0m[1maws_iam_role_policy_attachment.ecs_task_execution_role: Creating...[0m[0m
[0m[1maws_ecs_task_definition.app: Creating...[0m[0m
[0m[1maws_vpc.main: Creation complete after 2s [id=vpc-0a22d1d4ed5fe5147][0m[0m
[0m[1maws_subnet.private[0]: Creating...[0m[0m
[0m[1maws_internet_gateway.gw: Creating...[0m[0m
```

Jenkins › MidSemester › #10

```
[0m[1maws_appautoscaling_policy.up: Creating...[0m[0m
[0m[1maws_appautoscaling_policy.down: Creating...[0m[0m
[0m[1maws_appautoscaling_policy.up: Creation complete after 0s [id=myapp_scale_up][0m
[0m
[0m[1maws_appautoscaling_policy.down: Creation complete after 0s
[id=myapp_scale_down][0m[0m
[0m[1maws_cloudwatch_metric_alarm.service_cpu_high: Creating...[0m[0m
[0m[1maws_cloudwatch_metric_alarm.service_cpu_low: Creating...[0m[0m
[0m[1maws_cloudwatch_metric_alarm.service_cpu_high: Creation complete after 1s
[id=myapp_cpu_utilization_high][0m[0m
[0m[1maws_cloudwatch_metric_alarm.service_cpu_low: Creation complete after 1s
[id=myapp_cpu_utilization_low][0m[0m
[0m[1m[32m
Apply complete! Resources: 32 added, 0 changed, 0 destroyed.[0m
[0m[1m[32m
Outputs:

alb_hostname = myapp-load-balancer-2082785267.ap-south-1.elb.amazonaws.com[0m
Finished: SUCCESS
```

That’s it. We have succesfully hosted our MERN Application in containerized form on the AWS Fargate cluster.