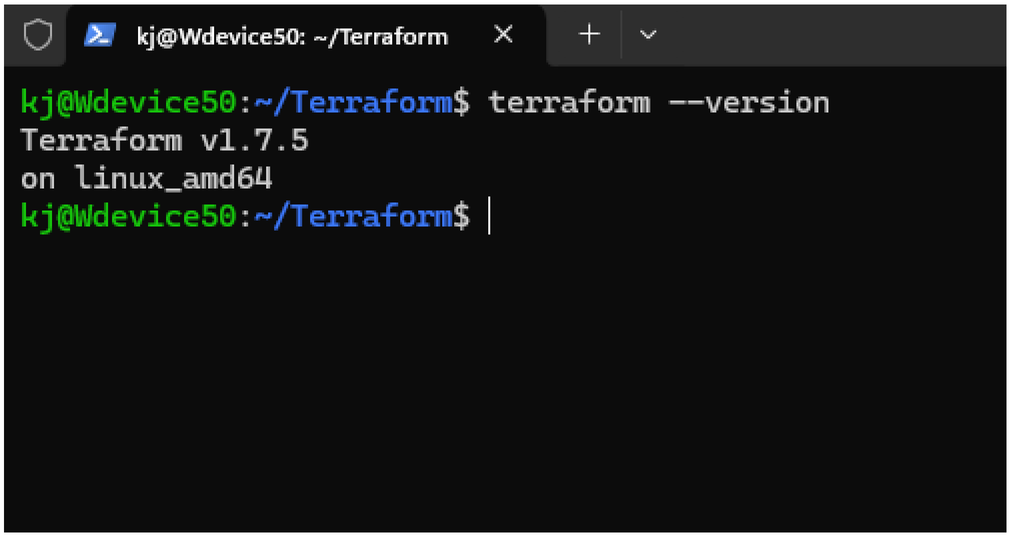
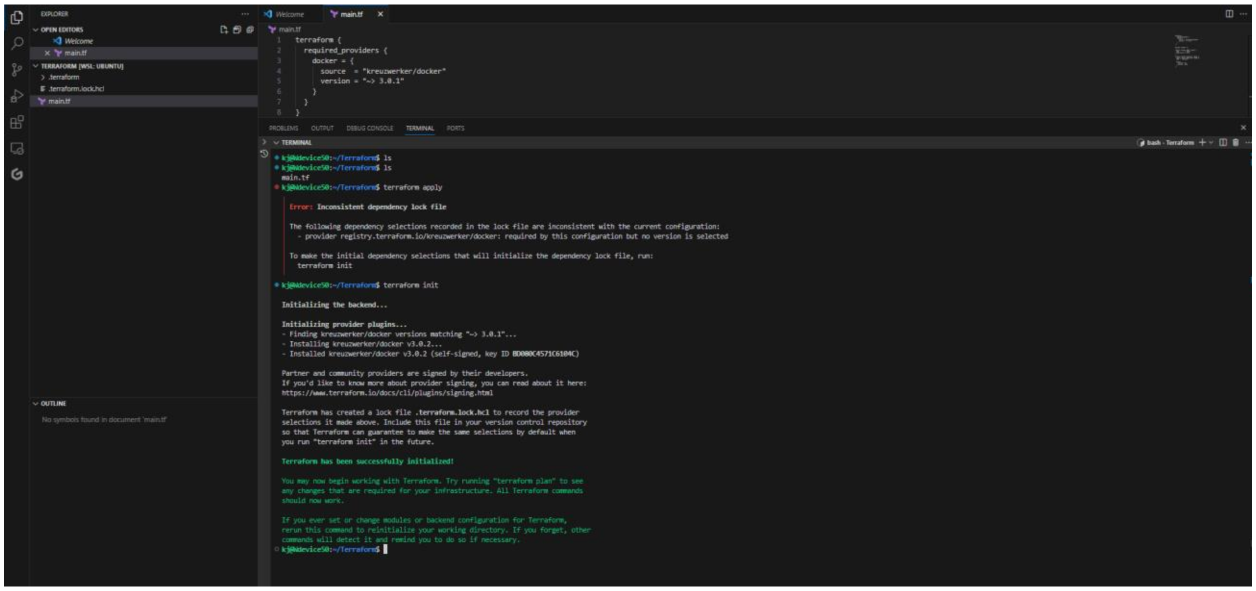
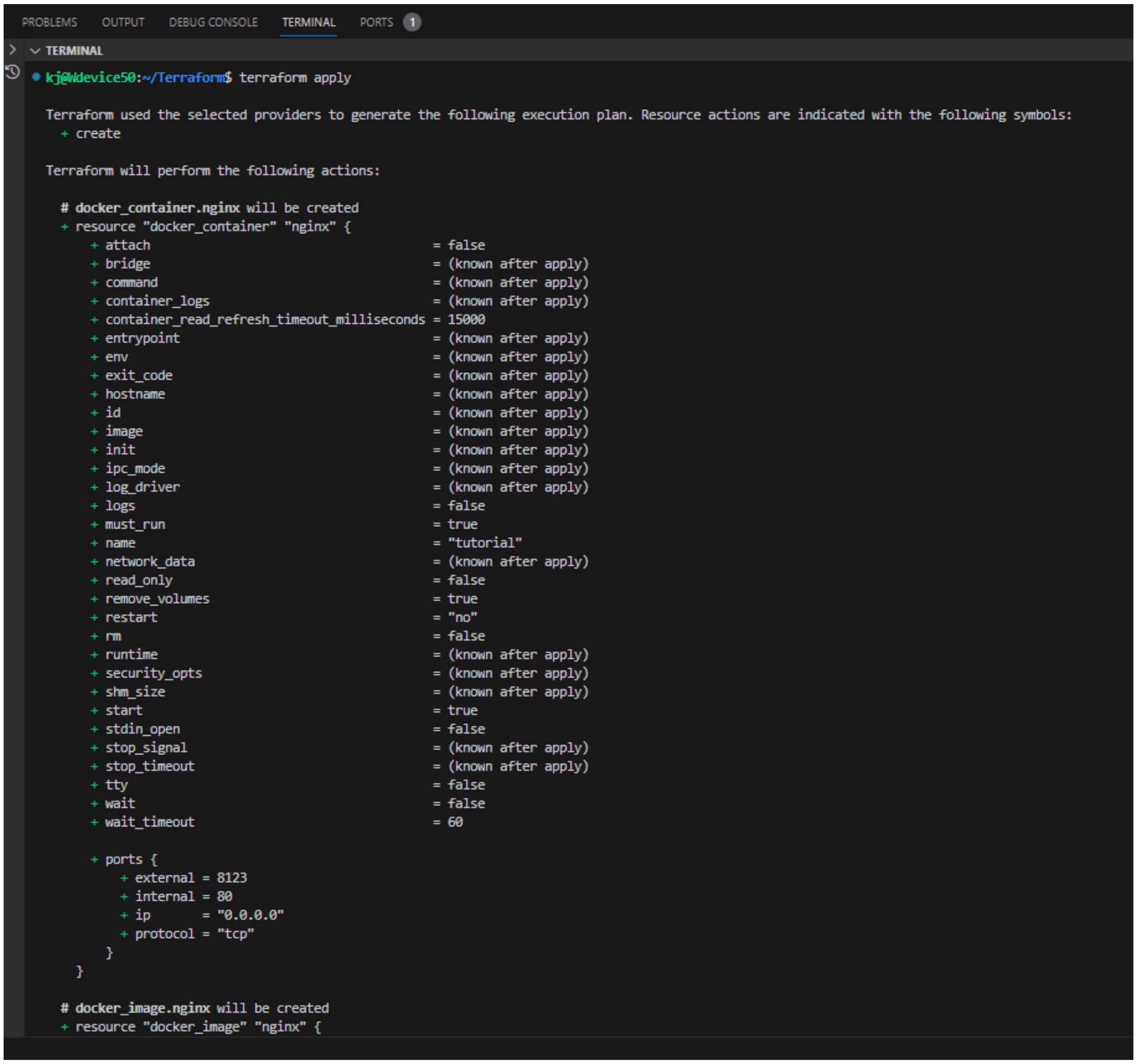


```
kj@Wdevice50:~/Terraform$ terraform --version
Terraform v1.7.5
on linux_amd64
kj@Wdevice50:~/Terraform$
```

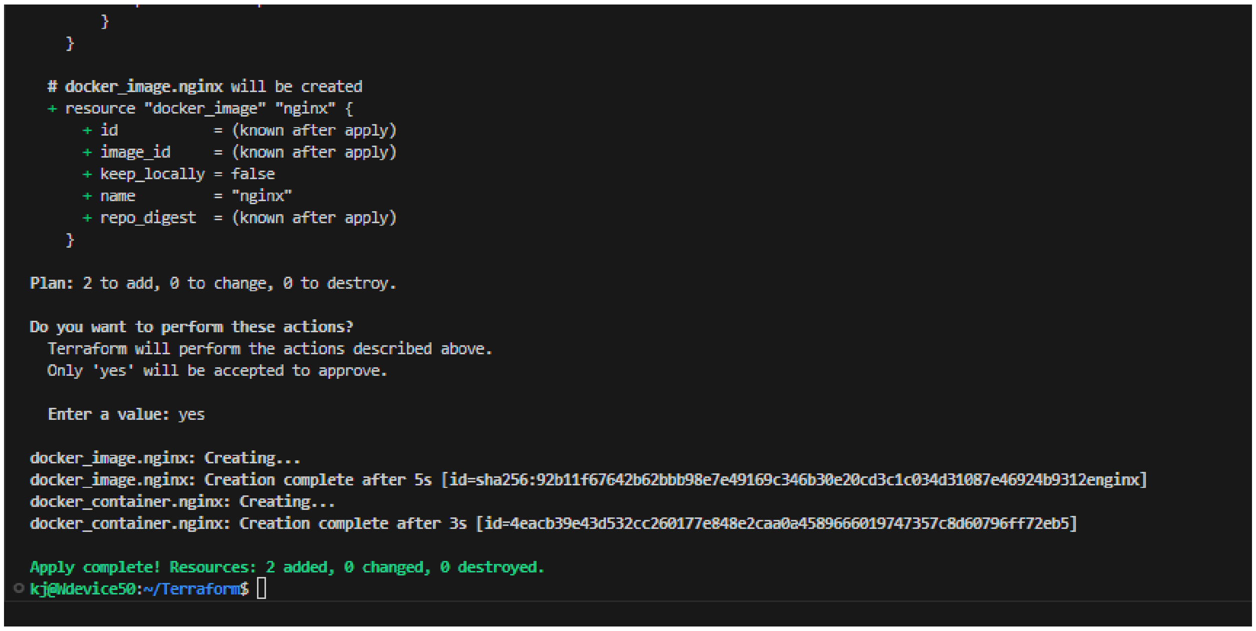


```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}
```

```
kj@Wdevice50:~/Terraform$ ls
kj@Wdevice50:~/Terraform$ ls
main.tf
kj@Wdevice50:~/Terraform$ terraform apply

  Error: Inconsistent dependency lock file

  The following dependency selections recorded in the lock file are inconsistent with the current configuration:
    - provider registry.terraform.io/kreuzwerker/docker: required by this configuration but no version is selected

  To make the initial dependency selections that will initialize the dependency lock file, run:
    terraform init

kj@Wdevice50:~/Terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "~> 3.0.1"...
- Installing kreuzwerker/docker v3.0.2...
- Installed kreuzwerker/docker v3.0.2 (self-signed, key ID BD080C4571C6104C)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
kj@Wdevice50:~/Terraform$
```

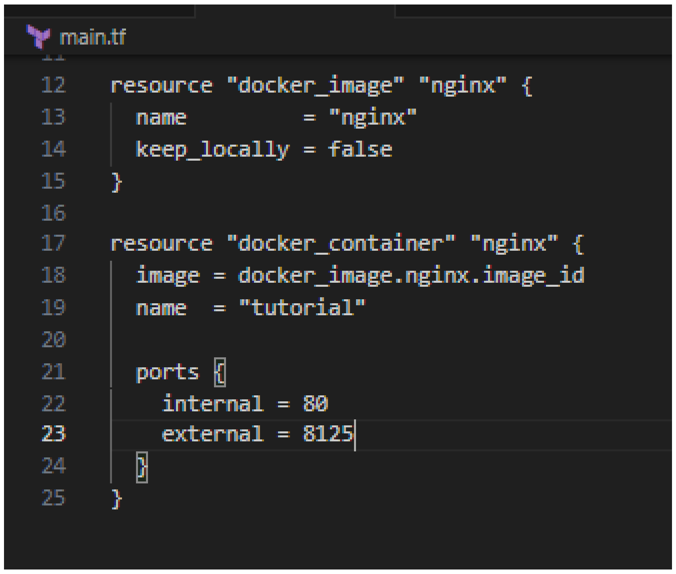


```
kj@Wdevice50:~/Terraform$ terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # docker_container.nginx will be created
  + resource "docker_container" "nginx" {
      + attach                                      = false
      + bridge                                      = (known after apply)
      + command                                     = (known after apply)
      + container_logs                              = (known after apply)
      + container_read_refresh_timeout_milliseconds = 15000
      + entrypoint                                  = (known after apply)
      + env                                         = (known after apply)
      + exit_code                                   = (known after apply)
      + hostname                                    = (known after apply)
      + id                                          = (known after apply)
      + image                                       = (known after apply)
      + init                                        = (known after apply)
      + ipc_mode                                    = (known after apply)
      + log_driver                                  = (known after apply)
      + logs                                        = false
      + must_run                                    = true
      + name                                        = "tutorial"
      + network_data                                = (known after apply)
      + read_only                                   = false
      + remove_volumes                              = true
      + restart                                     = "no"
      + rm                                          = false
      + runtime                                     = (known after apply)
      + security_opts                               = (known after apply)
      + shm_size                                    = (known after apply)
      + start                                       = true
      + stdin_open                                  = false
      + stop_signal                                 = (known after apply)
      + stop_timeout                                = (known after apply)
      + tty                                         = false
      + wait                                        = false
      + wait_timeout                                = 60

      + ports {
          + external = 8123
          + internal = 80
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_image.nginx will be created
  + resource "docker_image" "nginx" {
```



```
        }
    }

  # docker_image.nginx will be created
  + resource "docker_image" "nginx" {
      + id           = (known after apply)
      + image_id     = (known after apply)
      + keep_locally = false
      + name         = "nginx"
      + repo_digest  = (known after apply)
    }

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

docker_image.nginx: Creating...
docker_image.nginx: Creation complete after 5s [id=sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312enginx]
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 3s [id=4eacb39e43d532cc260177e848e2caa0a4589666019747357c8d60796ff72eb5]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
kj@Wdevice50:~/Terraform$ 
```

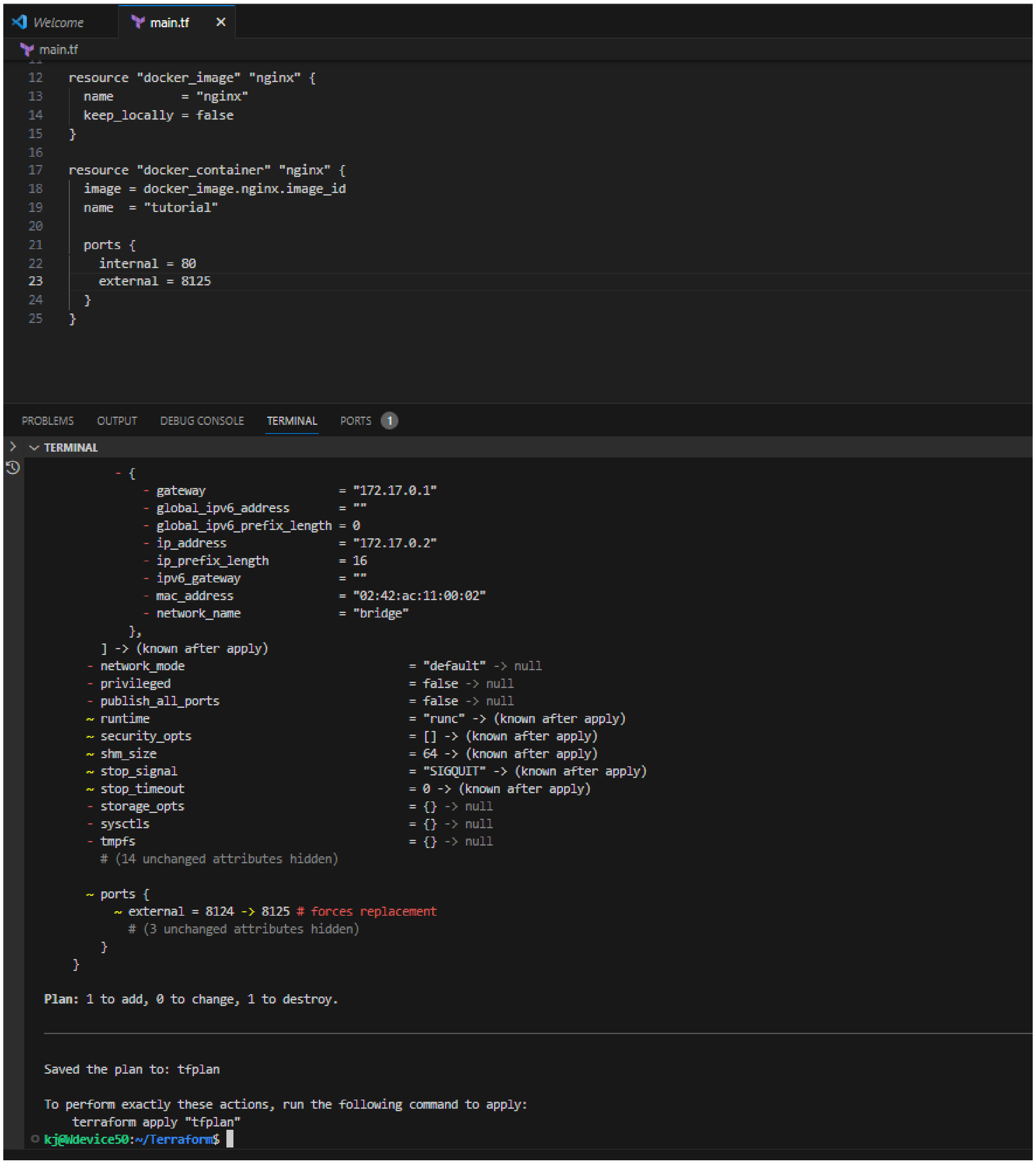
-Change infrastructure (your choice of what change it is.)



main.tf

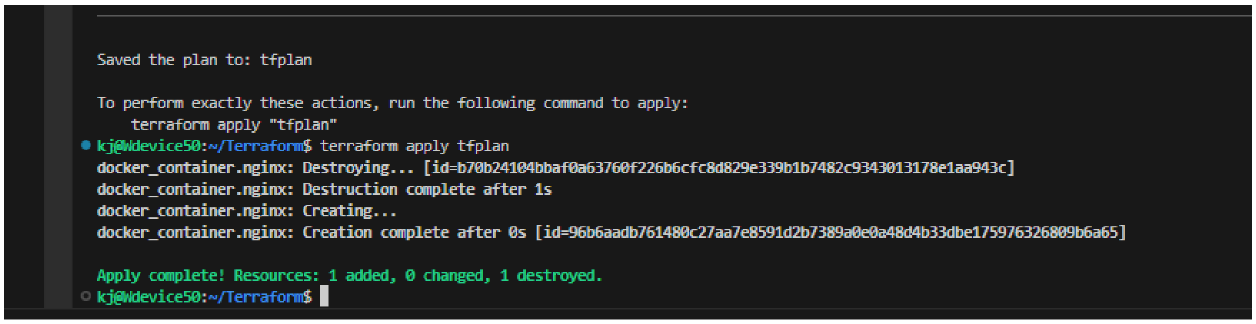
```
resource "docker_image" "nginx" {
  name         = "nginx"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"

  ports {
    internal = 80
    external = 8125
  }
}
```

-Create a plan and use the plan to make changes to the resource.



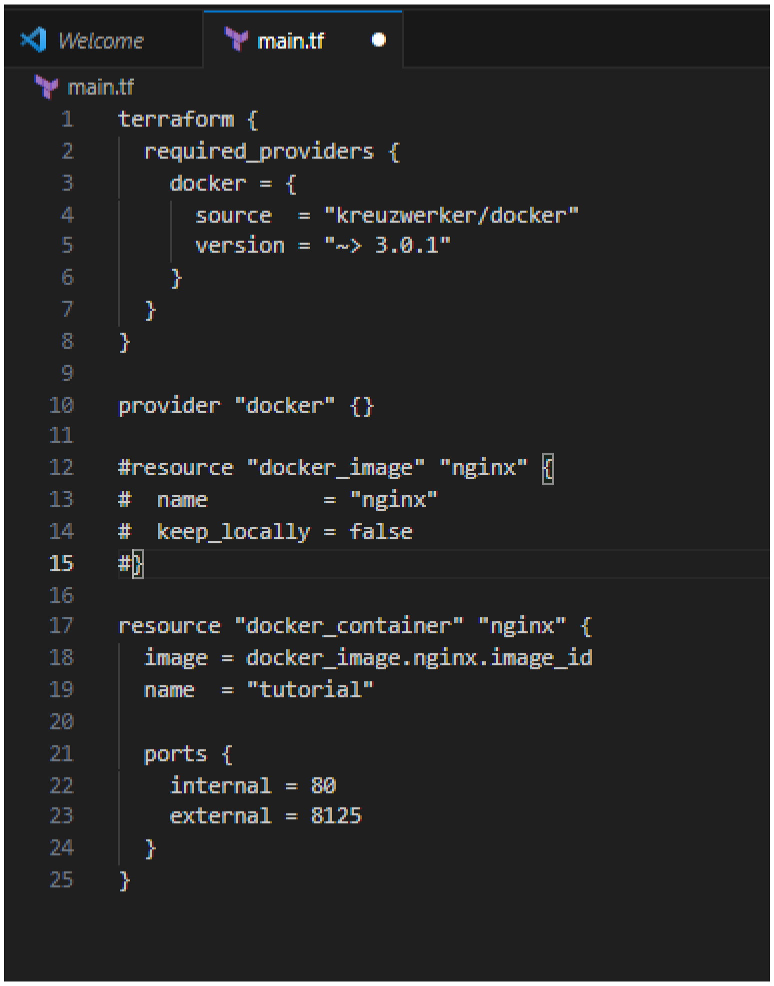
```
resource "docker_image" "nginx" {
  name         = "nginx"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"

  ports {
    internal = 80
    external = 8125
  }
}
```

```
          - {
              - gateway                   = "172.17.0.1"
              - global_ipv6_address       = ""
              - global_ipv6_prefix_length = 0
              - ip_address                = "172.17.0.2"
              - ip_prefix_length          = 16
              - ipv6_gateway              = ""
              - mac_address               = "02:42:ac:11:00:02"
              - network_name              = "bridge"
            },
        ] -> (known after apply)
      - network_mode                                = "default" -> null
      - privileged                                  = false -> null
      - publish_all_ports                           = false -> null
      ~ runtime                                     = "runc" -> (known after apply)
      ~ security_opts                               = [] -> (known after apply)
      ~ shm_size                                    = 64 -> (known after apply)
      ~ stop_signal                                 = "SIGQUIT" -> (known after apply)
      ~ stop_timeout                                = 0 -> (known after apply)
      - storage_opts                                = {} -> null
      - sysctls                                     = {} -> null
      - tmpfs                                       = {} -> null
        # (14 unchanged attributes hidden)

      ~ ports {
          ~ external = 8124 -> 8125 # forces replacement
            # (3 unchanged attributes hidden)
        }
    }

Plan: 1 to add, 0 to change, 1 to destroy.

─────────────────────────────────────────────────────────────────────────────

Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
kj@Wdevice50:~/Terraform$ 
```

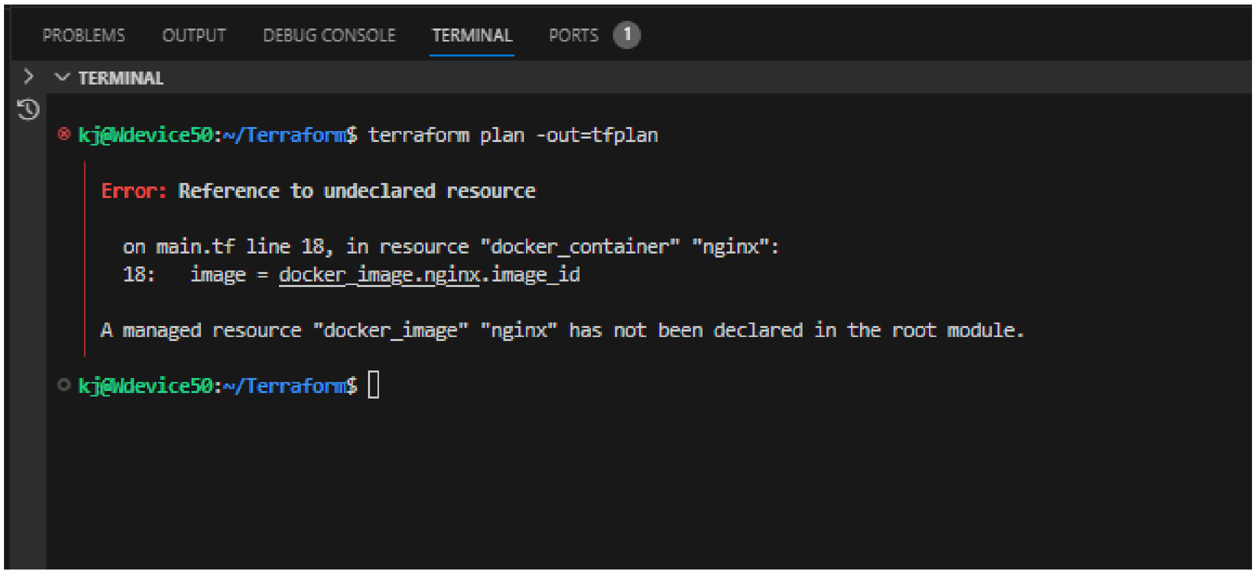


```
Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
kj@Wdevice50:~/Terraform$ terraform apply tfplan
docker_container.nginx: Destroying... [id=b70b24104bbaf0a63760f226b6cfc8d829e339b1b7482c9343013178e1aa943c]
docker_container.nginx: Destruction complete after 1s
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 0s [id=96b6aadb761480c27aa7e8591d2b7389a0e0a48d4b33dbe175976326809b6a65]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
kj@Wdevice50:~/Terraform$
```

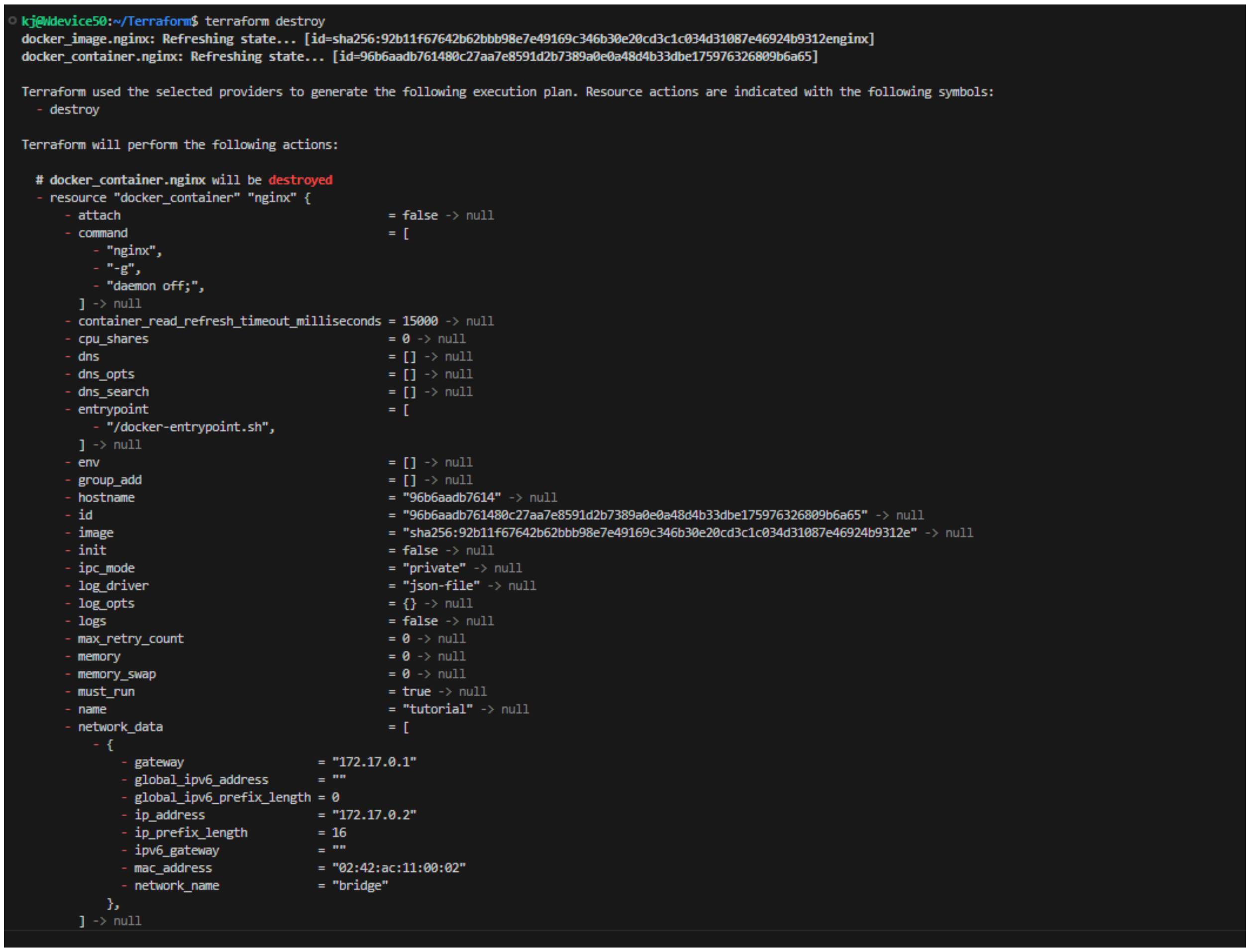
-Make destructive changes (like removing one of the docker images.)



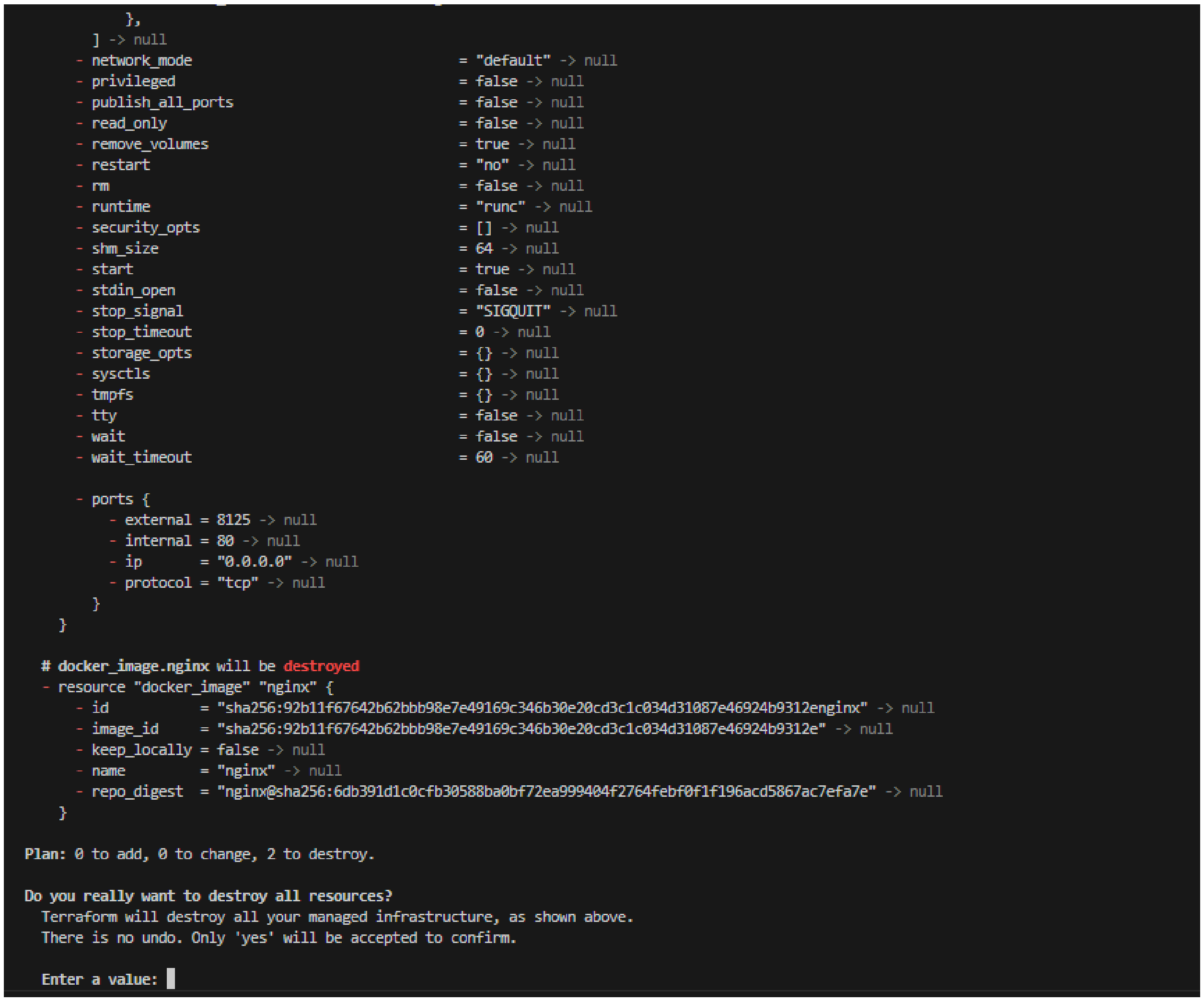
```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}

provider "docker" {}

#resource "docker_image" "nginx" {
#  name         = "nginx"
#  keep_locally = false
#}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"

  ports {
    internal = 80
    external = 8125
  }
}
```



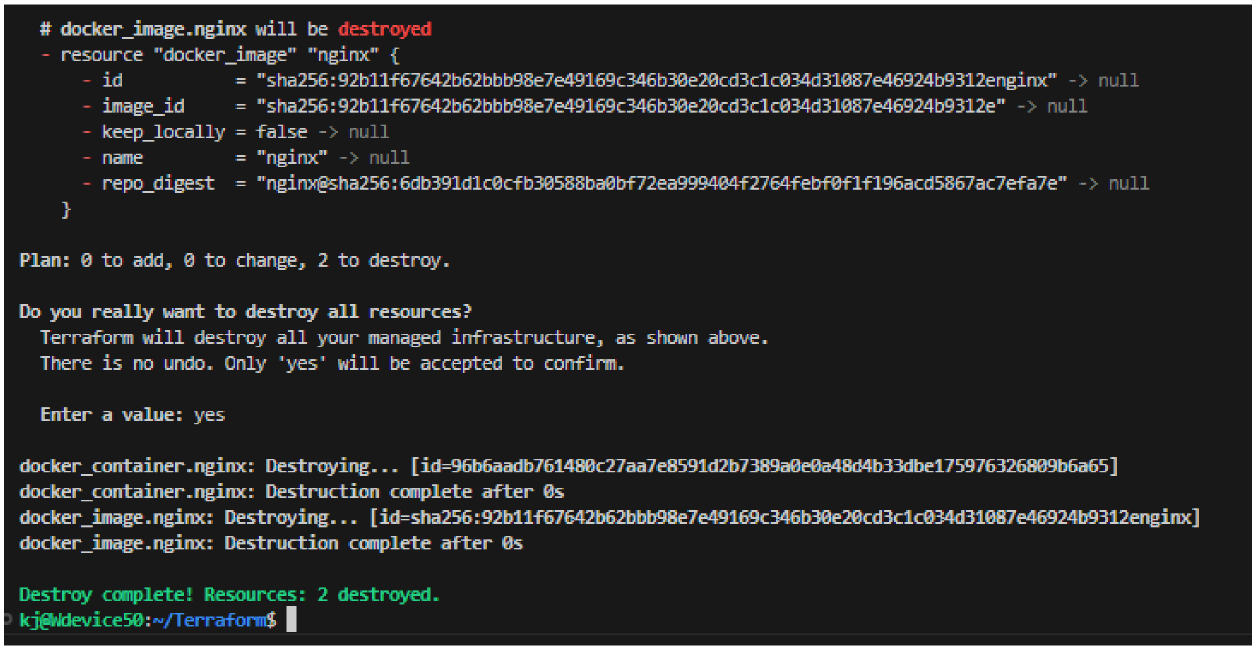
```
⊗ kj@Wdevice50:~/Terraform$ terraform plan -out=tfplan

  Error: Reference to undeclared resource

    on main.tf line 18, in resource "docker_container" "nginx":
    18:    image = docker_image.nginx.image_id

  A managed resource "docker_image" "nginx" has not been declared in the root module.

○ kj@Wdevice50:~/Terraform$
```



```
○ kj@Wdevice50:~/Terraform$ terraform destroy
  docker_image.nginx: Refreshing state... [id=sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312enginx]
  docker_container.nginx: Refreshing state... [id=96b6aadb761480c27aa7e8591d2b7389a0e0a48d4b33dbe175976326809b6a65]

  Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
    - destroy

  Terraform will perform the following actions:

    # docker_container.nginx will be destroyed
    - resource "docker_container" "nginx" {
        - attach                                      = false -> null
        - command                                     = [
            - "nginx",
            - "-g",
            - "daemon off;",
          ] -> null
        - container_read_refresh_timeout_milliseconds = 15000 -> null
        - cpu_shares                                  = 0 -> null
        - dns                                         = [] -> null
        - dns_opts                                    = [] -> null
        - dns_search                                  = [] -> null
        - entrypoint                                  = [
            - "/docker-entrypoint.sh",
          ] -> null
        - env                                         = [] -> null
        - group_add                                   = [] -> null
        - hostname                                    = "96b6aadb7614" -> null
        - id                                          = "96b6aadb761480c27aa7e8591d2b7389a0e0a48d4b33dbe175976326809b6a65" -> null
        - image                                       = "sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312e" -> null
        - init                                        = false -> null
        - ipc_mode                                    = "private" -> null
        - log_driver                                  = "json-file" -> null
        - log_opts                                    = {} -> null
        - logs                                        = false -> null
        - max_retry_count                             = 0 -> null
        - memory                                      = 0 -> null
        - memory_swap                                 = 0 -> null
        - must_run                                    = true -> null
        - name                                        = "tutorial" -> null
        - network_data                                = [
            - {
                - gateway                   = "172.17.0.1"
                - global_ipv6_address       = ""
                - global_ipv6_prefix_length = 0
                - ip_address                = "172.17.0.2"
                - ip_prefix_length          = 16
                - ipv6_gateway              = ""
                - mac_address               = "02:42:ac:11:00:02"
                - network_name              = "bridge"
              },
          ] -> null
```

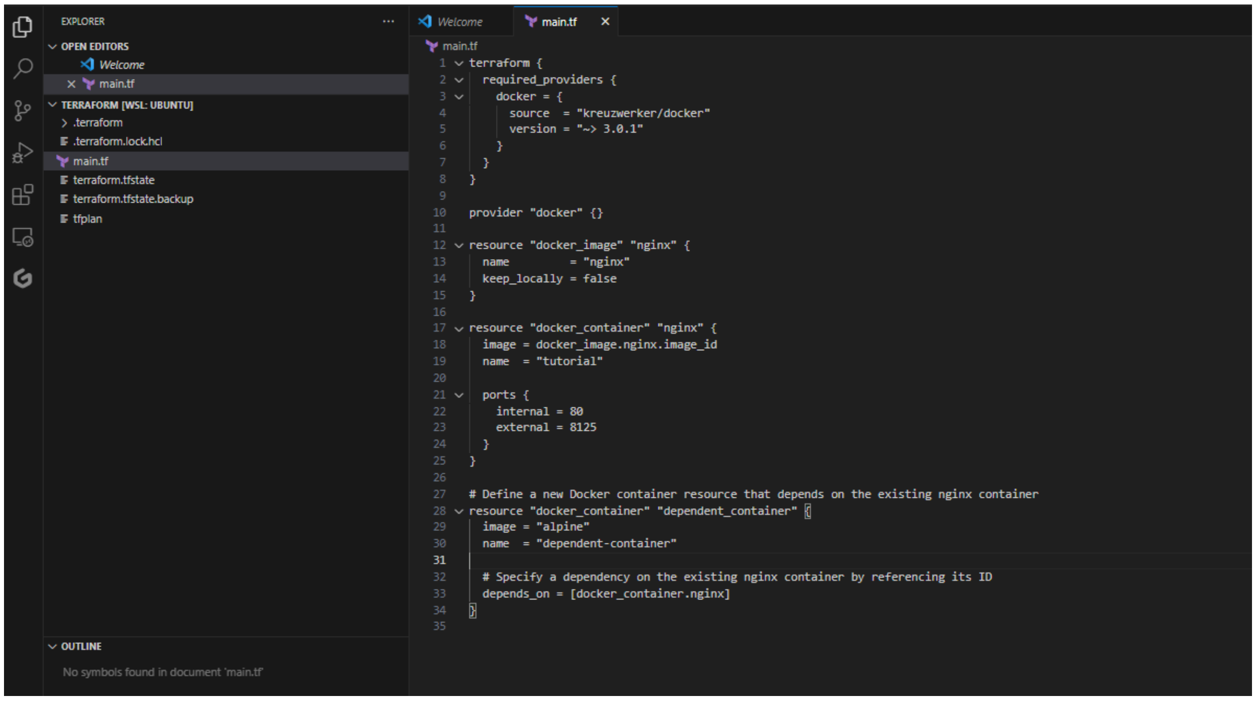


```
            },
        ] -> null
      - network_mode                                = "default" -> null
      - privileged                                  = false -> null
      - publish_all_ports                           = false -> null
      - read_only                                   = false -> null
      - remove_volumes                              = true -> null
      - restart                                     = "no" -> null
      - rm                                          = false -> null
      - runtime                                     = "runc" -> null
      - security_opts                               = [] -> null
      - shm_size                                    = 64 -> null
      - start                                       = true -> null
      - stdin_open                                  = false -> null
      - stop_signal                                 = "SIGQUIT" -> null
      - stop_timeout                                = 0 -> null
      - storage_opts                                = {} -> null
      - sysctls                                     = {} -> null
      - tmpfs                                       = {} -> null
      - tty                                         = false -> null
      - wait                                        = false -> null
      - wait_timeout                                = 60 -> null

      - ports {
          - external = 8125 -> null
          - internal = 80 -> null
          - ip       = "0.0.0.0" -> null
          - protocol = "tcp" -> null
        }
    }

  # docker_image.nginx will be destroyed
  - resource "docker_image" "nginx" {
      - id           = "sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312enginx" -> null
      - image_id     = "sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312e" -> null
      - keep_locally = false -> null
      - name         = "nginx" -> null
      - repo_digest  = "nginx@sha256:6db391d1c0cfb30588ba0bf72ea999404f2764febf0f1f196acd5867ac7efa7e" -> null
    }

Plan: 0 to add, 0 to change, 2 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: 
```



```
  # docker_image.nginx will be destroyed
  - resource "docker_image" "nginx" {
      - id           = "sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312enginx" -> null
      - image_id     = "sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312e" -> null
      - keep_locally = false -> null
      - name         = "nginx" -> null
      - repo_digest  = "nginx@sha256:6db391d1c0cfb30588ba0bf72ea999404f2764febf0f1f196acd5867ac7efa7e" -> null
    }

Plan: 0 to add, 0 to change, 2 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

docker_container.nginx: Destroying... [id=96b6aadb761480c27aa7e8591d2b7389a0e0a48d4b33dbe175976326809b6a65]
docker_container.nginx: Destruction complete after 0s
docker_image.nginx: Destroying... [id=sha256:92b11f67642b62bbb98e7e49169c346b30e20cd3c1c034d31087e46924b9312enginx]
docker_image.nginx: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
kj@Wdevice50:~/Terraform$ 
```

-Create resource with dependencies (implicit and explicit)



```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}

provider "docker" {}

resource "docker_image" "nginx" {
  name         = "nginx"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"

  ports {
    internal = 80
    external = 8125
  }
}

# Define a new Docker container resource that depends on the existing nginx container
resource "docker_container" "dependent_container" {
  image = "alpine"
  name  = "dependent-container"

  # Specify a dependency on the existing nginx container by referencing its ID
  depends_on = [docker_container.nginx]
}
```

Use cases for each task

-Install Terraform in your local system.

Install Terraform

-Create a tf configuration for running docker resource. You can take help from this tutorial.

Deploying a Microservices Architecture

Imagine you're tasked with deploying a microservices-based application for a retail e-commerce platform. The application consists of several microservices, each running in a Docker container, and you want to use Terraform to manage the deployment of these containers on a cloud platform.

-Apply the initial resources.

Apply resources

-Change infrastructure (your choice of what change it is.)

Updating Firewall Rules

Imagine you're responsible for managing the infrastructure for a financial services company. As part of your organization's security policies, you need to update the firewall rules to restrict access to certain services and ports.

-Create a plan and use the plan to make changes to the resource.

Scaling Web Application Infrastructure: Imagine you're managing the infrastructure for a popular e-commerce website that experiences spikes in traffic during holiday seasons or special promotions. To handle increased demand, you need to scale your infrastructure dynamically while maintaining high availability and performance.

-Make destructive changes (like removing one of the docker images.)

Application Updates: When updating your application's dependencies or components, you may need to remove old or unused Docker images to free up disk space and ensure that only the necessary images are available for deployment.

-Destroy the complete resource.

Environment Cleanup: When decommissioning an environment, such as a development or testing environment, you may want to remove all resources associated with that environment to free up resources and reduce costs. This ensures that no residual resources are left running unnecessarily.

-Create resource with dependencies (implicit and explicit)

Imagine you have a microservices architecture where each service runs in its own Docker container. Your application consists of several services, including a web server (nginx), a database service (MySQL), and an API service (Node.js). These services may have dependencies on each other, and you want to manage them using Terraform.

Here's how you could use the configuration:

1. **nginx Container**:
    - The nginx container serves as the front-end web server, handling HTTP requests from clients. It serves static content and forwards API requests to the API service.
2. **API Service Container**:
    - The API service container runs a Node.js application that provides the application's backend functionality. It handles API requests from clients, interacts with the database service, and performs business logic.
3. **MySQL Database Container**:
    - The MySQL container hosts the application's database. It stores data used by the API service and allows the API service to persist and retrieve data.

In this scenario, the nginx container depends on the API service container to handle API requests, and the API service container depends on the MySQL container to store and retrieve data. By defining these dependencies in your Terraform configuration, you ensure that containers are created and started in the correct order, avoiding issues related to missing dependencies or race conditions.