Changes Made to Docker file for Improved Code Quality and Best Practices:

In below I am explaining the changes made by me to given docker file to achieve code quality and best practices and future improvements section explains how to achieve security and automation.

```
# Use a specific version tag for the base image
FROM python:3.8-slim AS base

# Set the working directory
WORKDIR /usr/src/app

# Copy only the requirements file first
COPY requirements.txt ./

# Create a non-root user and switch to it
RUN adduser  myuser
USER myuser

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Use a separate stage for the application
FROM base AS app

# Copy the application code
COPY . .

# Specify the entrypoint command
ENTRYPOINT ["python", "./main.py"]
```

- This Docker file uses multi-stage builds to separate the installation of dependencies from the application code, resulting in a smaller and more efficient final image.
- Provided a clear documentation within the Docker file to explain each step and its purpose. This can help other developers understand the configuration and make it easier to maintain in the future.
- **It's generally a best practice to avoid running commands as the root user whenever possible to minimize security risks and potential conflicts. By creating a non-root user (my user in this example) and switching to it before running pip install, you can mitigate potential issues related to permissions and conflicting behaviour with the system package manager.**

Future Improvements: -

1. Automation:

Set up a continuous integration/continuous deployment (CI/CD) pipeline to automate the building and deployment of Docker images. This can help streamline workflows and improve efficiency by reducing manual intervention.

2. Security:

Consider scanning the Docker image for vulnerabilities using tools like Clair or Trivy as part of your CI/CD pipeline to identify and address any security issues.

Changes Made to Workflow file for Improved Code Quality and Best Practices:

In below I am explaining the changes made by me to given create-repo.yaml to achieve code quality and best practices and future improvements section explains how to achieve security and automation.

```yaml
name: Create Repository

on:
  workflow_dispatch:
    inputs:
      name:
        description: "Repository Name"
        required: true
        type: string
      organization:
        description: "Organization Name"
        required: true
        type: string
      description:
        description: "Repository Description"
        required: false
        type: string
      engineer-type:
        description: "Engineer Type (platform or data)"
        required: true
        type: string
      defbranch:
        description: "Default Branch"
        required: false
        default: "main"

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Check out code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build and run Docker image
        run: |
          docker build -t repo-creator .
          docker run --env GITHUB_TOKEN=${{ secrets.GH_TOKEN }} repo-creator -
n ${{ github.event.inputs.name }} -o ${{ github.event.inputs.organization }} -
```

```
d "${{ github.event.inputs.description }}" -b ${{
github.event.inputs.defbranch }} -t ${{ github.event.inputs.engineer-type }}

  lint:
    runs-on: ubuntu-latest

    steps:
      - name: Check out code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: 3.x

      - name: Install dependencies
        run: pip install flake8

      - name: Lint Python code
        run: flake8 main.py
```

- I have created a input variable **engineer-type** based on the value it will create repository for platform/data engineers.
- By specifying the data type for each input (e.g., string, number, boolean), you make it clear to users what kind of data is expected for each input field. This improves the readability of the workflow file and reduces the likelihood of errors or misunderstandings when invoking the workflow.
- The linting job (lint)is configured to run on the ubuntu-latest environment. The steps in the linting job first check out the code, set up Python, install Flake8, and then run Flake8 to lint the main.py file. I am triggering the linting job separately from the build job.
- Linting can identify syntax errors, potential bugs, and style violations in our codebase. Running linting as part of your CI workflow ensures that these issues are detected early on, before they have a chance to propagate further downstream.

Future Improvements: -

1. **Private Runners: -**

In above GitHub Actions workflow file, the job is configured to run on a GitHub-hosted runner, which is a shared infrastructure provided by GitHub. The line runs-on: ubuntu-latest specifies that the job will run on an Ubuntu virtual machine provided by GitHub as part of the shared runner pool. However, it's important to note that while shared runners are convenient, they might have limitations in terms of resource availability and performance. Private runners offer several advantages, especially for organizations with specific security, compliance, or performance requirements.

2. **Security:**

   Implement scanning tools to detect and remove hardcoded secrets from your codebase. Tools like GitGuardian can help identify and remove sensitive information accidentally committed to your repository. Enable branch protection rules to prevent direct commits to protected branches and require pull requests for code review. This helps ensure that changes are reviewed before they are merged, reducing the risk of introducing security vulnerabilities.

Changes Made to Main.py to achieve actual functionality.

- I have implemented two user defined functions with name configure_for_platform_engineer responsible for creating repository for platform engineers in GitHub with default template and configure_for_data_engineer responsible for creating repository for data engineers in GitHub with default template.
- I write conditional block based on user input if user selects data, then repository for data engineers in GitHub with default template is created. User selects platform repository for platform engineers in GitHub with default template is created.

Best practices:

1. Implement exception handling to gracefully handle errors and failures in API requests.
2. implementing the add_terraform_configurations and add_python_configurations methods with actual configurations to adhere to best practices for Terraform and Python projects.
3. Include comments throughout the code to explain complex logic or implementation details.

Future Improvements: -

- Current we have only integrated lint only on CI/CD pipeline in future improvements Implement CI/CD configurations to automate testing, building, and deploying the repository.
- Integrate Security scan tool to CI/CD tool to identify vulnerabilities in code .