OPA - docker

Open Policy Agent, also known as OPA, is a popular open source policy engine that provides a unified policy language and framework that can be used across different layers of the technology stack. The OPA project was started in 2016 by Styra, Inc. and has quickly gained popularity in the cloud-native and DevOps communities.
In December 2020, the Open Policy Agent project announced that it was becoming a part of the Cloud Native Computing Foundation (CNCF), the leading open source organization that hosts and governs many popular cloud-native projects such as Kubernetes, Prometheus, and Envoy.
As a CNCF-hosted project, OPA will continue to evolve and innovate with the help of the larger open source community, while also benefiting from the governance and organizational support provided by the CNCF.
Overall, the Open Policy Agent Foundation represents an important step forward in the development and adoption of policy-as-code approaches for governing modern cloud-native and DevOps environments.

OPA.REGO file:

```
package main

# Do Not store secrets in ENV variables
secrets_env = [
   "passwd",
   "password",
   "pass",
   "secret",
   "key",
   "access",
   "api_key",
   "apikey",
   "token",
   "tkn"
]

deny[msg] {
   input[i].Cmd == "env"
   val := input[i].Value
   contains(lower(val[_]), secrets_env[_])
   msg = sprintf("Line %d: Potential secret in ENV key found: %s", [i, val])
}

# Only use trusted base images
deny[msg] {
   input[i].Cmd == "from"
   val := split(input[i].Value[0], "/")
   count(val) > 1
   msg = sprintf("Line %d: use a trusted base image", [i])
}

# Do not use 'latest' tag for base imagedeny[msg] {
deny[msg] {
   input[i].Cmd == "from"
```

```
    val := split(input[i].Value[0], ":")
    contains(lower(val[1]), "latest")
    msg = sprintf("Line %d: do not use 'latest' tag for base images", [i])
}

# Avoid curl bashing
deny[msg] {
    input[i].Cmd == "run"
    val := concat(" ", input[i].Value)
    matches := regex.find_n("(curl|wget)[^|^>]*[|>]", lower(val), -1)
    count(matches) > 0
    msg = sprintf("Line %d: Avoid curl bashing", [i])
}

# Do not upgrade your system packages
warn[msg] {
    input[i].Cmd == "run"
    val := concat(" ", input[i].Value)
    matches := regex.match(".*?(apk|yum|dnf|apt|pip).+?(install|[dist-|check-|group]?up[grade|date]).*", lower(val))
    matches == true
    msg = sprintf("Line: %d: Do not upgrade your system packages: %s", [i, val])
}

# Do not use ADD if possible
deny[msg] {
    input[i].Cmd == "add"
    msg = sprintf("Line %d: Use COPY instead of ADD", [i])
}

# Any user...
any_user {
    input[i].Cmd == "user"
 }

deny[msg] {
    not any_user
    msg = "Do not run as root, use USER instead"
}

# ... but do not root
forbidden_users = [
    "root",
    "toor",
    "0"
]

deny[msg] {
    command := "user"
    users := [name | input[i].Cmd == command; name := input[i].Value]
    lastuser := users[count(users)-1]
    contains(lower(lastuser[_]), forbidden_users[_])
    msg = sprintf("Line %d: Last USER directive (USER %s) is forbidden", [input.Line, lastuser])
}
```

```
# Do not sudo
deny[msg] {
   input[i].Cmd == "run"
   val := concat(" ", input[i].Value)
   contains(lower(val), "sudo")
   msg = sprintf("Line %d: Do not use 'sudo' command", [i])
}

# Use multi-stage builds
default multi_stage = false
multi_stage = true {
   input[i].Cmd == "copy"
   val := concat(" ", input[i].Flags)
   contains(lower(val), "--from=")
}
deny[msg] {
   multi_stage == false
   msg = sprintf("You COPY, but do not appear to use multi-stage builds...", [])
}
```

The first rule prohibits storing secrets in environment variables. It will trigger a deny message if any of the specified strings (i.e., "passwd", "password", "pass", etc.) are found in an `**ENV**` key.
The second rule ensures that only trusted base images are used. It reports any Dockerfiles that use base images from sources that are not in the form of a Docker registry.
The third rule prohibits the use of the "latest" tag for base images.
The fourth rule alert any use curl in a docker container.
The fifth rule warns against upgrading system packages in Dockerfiles.
The sixth rule prohibits the use of the `**ADD**` command in a Dockerfile to copy files, and recommends using the `**COPY**` command instead.
The seventh rule warns against running Docker containers as root, and suggests setting the `**USER**` directive.
The eighth rule prohibits the use of the `**sudo**` command in a Dockerfile.
The ninth rule recommends the use of multi-stage Docker builds, and warns if there are any COPY commands that appear outside of the build's multi-stage.
In summary, this Rego file defines rules to enforce Docker security best practices, and it can be used with OPA to prevent vulnerabilities in a Dockerized service or application.

Reference dockerfile best practices:
https://cloudberry.engineering/article/dockerfile-security-best-practices/
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Add opa Conftest

```
stage('Vulnerability Scan'){
     steps {
       parallel(
         "Dependency Scan": {
          sh "mvn dependency-check:check"
        },
        "Trivy Scan": {
          sh "bash trivy-scan1.sh"
        },
        "OPA Conftest": {
```

```
        sh 'docker run --rm -v $(pwd):/project openpolicyagent/conftest test --policy opa-docker-security2.rego Dockerfile'
      }
    )
  }
}
```

The `OPA Conftest` step runs a container image scan using the Open Policy Agent conftest tool by running the container with a bind mount volume to include the project root directory in container and then testing the Dockerfile against the `opa-docker-security2.rego` policy using the command `docker run --rm -v $(pwd):/project openpolicyagent/conftest test --policy opa-docker-security2.rego Dockerfile`.

Now run build and it is will fail with result below:

```
+ docker run --rm -v /var/lib/jenkins/workspace/devsecops-app-num:/project openpolicyagent/conftest test --policy opa-docker-security2.rego Dockerfile
FAIL- Dockerfile - main - Do not run as root, use USER instead
FAIL- Dockerfile - main - Line 0: use a trusted base image
FAIL- Dockerfile - main - Line 3: Use COPY instead of ADD
FAIL- Dockerfile - main - You COPY, but do not appear to use multi-stage builds...

10 tests, 6 passed, 0 warnings, 4 failures, 0 exceptions
script returned exit code 1
```

We need edit Dockerfile to measure secure like below.

```
FROM adoptopenjdk/openjdk8:alpine-slim
EXPOSE 8080
ARG JAR_FILE=target/*.jar
RUN addgroup -S pipeline && adduser -S k8s-pipeline -G pipeline
COPY ${JAR_FILE} /home/k8s-pipeline/app.jar
USER k8s-pipeline
ENTRYPOINT ["java","-jar","/home/k8s-pipeline/app.jar"]
```

The Dockerfile mentioned follows some of the best security practices that should be included into the Dockerfile, which are:
Using minimal base images: In the given Dockerfile, the image has been used as "alpine-slim" which is a minimal and lightweight operating system designed for containers that ensures a smaller attack surface for the image by minimizing the potential vulnerabilities.
Specifying the user: The Dockerfile specifies the user "k8s-pipeline" instead of using the default root user. This practice makes the application running inside containers with limited access control to the host and promotes defense-in-depth security practices.
Making use of COPY instruction: The Dockerfile makes use of the COPY instruction so that the only contents copied to the container filesystem are the artifact needed to run the application (the JAR file). This offers better control over the image size and avoids unnecessary vulnerabilities.
Using ENTRYPOINT instruction: The ENTRYPOINT instruction is used to define the command that will be executed when the container starts. This ensures the correct command is executed and not any other commands that may be injected by an attacker.
Exposing only necessary ports: In the given Dockerfile, only port 8080 is exposed which is used by the running application. By exposing the only necessary port, we minimize the attack surface of our container.
By following these best practices, the Dockerfile ensures that only essential files are copied into the container, limits the need for privileges, and minimizes the attack surface of the container. This, in turn, makes the container more secure against potential attackers.