# **Part 2: Container Security implementation**

## **Docker Security Best Practices**

## **List and explain five Docker security best practices**

## **Implement one of these practices in a Dockerfile and provide the code.**

Answer:

Following are the Docker security best practices

1. Use trusted base images:

* Using trusted base images reduces the attach surface and potential vulnerabilities.
* Using trusted images from docker hub or other trusted sources.
* Implementing Docker image scan using different tools like synk for known vulnerabilities.
* Regularly patching the docker base image to incorporate latest security fix.

1. Least privilege
   * Application running inside the docker container should run with least privilege necessary to function.
   * Implementing dockers users’ flag and group flag for running container will certainly help.
2. Securing Docker Daemon and API
   * Network segmentation for any network docker daemon will be using.
   * Using TLS to encrypt the docker API communications.
   * Configuring Docker daemon to listen on Unix Socker or secured TCP socket.
3. Scanning vulnerabilities for Images
   * Implementing Docker images scan using different tools like synk for known vulnerabilities and patching the image as required.
4. Secrets Management:
   * Using Docker secrets for better management of sensitive data like passwords and tokens.
5. Logging and Monitoring:
   * Implementing better logging and monitoring for docker daemon, environment using IDS/IPS, as well as SIEM to detect and response to security incidents.

Dockerfile is created with the code is on same folder with name Dockerfile.

## **Kubernetes Security Configuration**

## **Describe three Kubernetes security features**

## **Write a Kubernetes YAML configuration that includes securityContext settings for a pod**

Answer:

Here are the Kubernetes security features

1. Namespaces:
   * + Namespace provides a way to divide the cluster resources between multiple applications, users or teams offering logical isolations within the cluster.
     + It can be used to isolate the environments too.
2. Pod Security Policies
   * We can define different security policies for the pods, so that it can be accepted by the system. This way, we can control the pods such as running as non-root users, use of host namespaces or restrict SELinux profiles etc.
3. Secrets Management
   * We can store secrets in base-64 encoded formats.
   * We can also securely mount the secrets in a pod as environment variables or volumes.
4. Network Policies
   * We can define different network policies based on the ingress and egress rules, so that we can control the traffic based on namespaces and labels.
5. RBAC Access
   * Set of rules can be defined based on different permissions and Role as well as Cluster binding roles.

Kubernetes YAML file with security context is in same folder with filename securityContext.yaml. Documenting the code here

This deploys and nginx-web pod with nginx:1.10.1-alpine as its docker image. As a securityContext, I have added nginx to run as user 1001 (uid) and gid as 1001. Also added extra security control to run as non-root user.

## **IaaS security Measures**

## **Explain the concept of Infrastructure as a Service (IaaS) and its security implications.**

Answer:

For using of Infrastructure as a Service, providers (Cloud providers) are responsible for providing the necessary networking, virtualized resources and on-demand workloads. IaaS providers manage the infrastructure, whereas as a customer, we have responsibility for operating systems, applications and its securities. Here are some of the security implications associated with IaaS

1. Shared responsibility Model:
   * On this model, provider is responsible for any underlying hardware security, whereas customers are responsible for securing everything that runs on top of that infrastructure, including, access control, data encryption and operating systems.
2. Protecting the data:
   * Customers are responsible for managing and protecting their own data. Customers’ needs to ensure the data encryption both at rest and transit. Customers are also responsible for their own data loss and leakage. Customers are responsible for data backups as well as comply with the local data policy and regulations.
3. Access:
   * Enabling Role base access control, Identity and Access Management to prevent unauthorized access is responsibility of the customers.
4. Monitor and Logging:
   * Its customer responsibility for monitoring and logging of network, applications and systems.
5. Compliance:
   * Customers are responsible for compliance and governance of the local data residing on particular area. They might need to follow the policies like GDPR, HIPAA, PCI-DSs etc.
   * Monitoring of those data and auditing is also responsibilities of the customers.
6. Patching
   * Customers are responsible for patching, vulnerability management, and threat modelling of their application and network they created in the infrastructure provided by customer.