IN GOD WE TRUST

DevSecOps Concepts

By

Alli-Darabi

Chapter one

Getting Started

Today DevOps allows organization to deploy changes faster than old methodologies, its set of Practices that combine the development and IT`s teams with aim of shortening software development life cycle with CI/CD practices.

From security point of view, DevOps could include a number of best practices that can be applied to increase the security of applications.

These best practices include following:

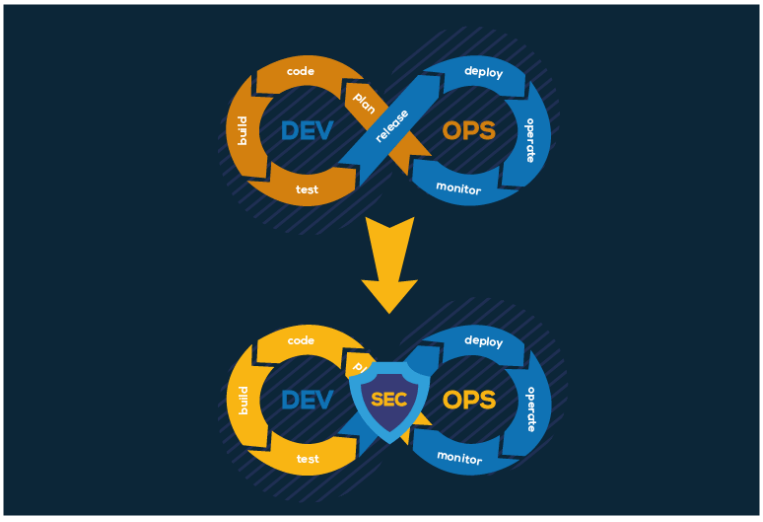
-Add automated security testing like FUZZ test and software penetration testing, to the software development lifecycle or system integration cycle.

-Standardization of the integration cycle to reduce the introduction of errors.

-Introduction of security issues and limitations to software and systems development teams at startup of the projects.

At this point, we can introduce DevSecOps as methodology that aims to integrate security tools into DevOps process in an automated way.

As a result, **this multicultural and multidisciplinary automated security environment makes security an issue that affects everyone and not just single team**! this is a main engine of DevSecOps.



Gartner provide a more precise definition of about DevSecOps:

"*DevSecOps is the integration of security in DevOps development in the most fluid and transparent way possible. ideally, this is done without reducing the agility or speed of developers or without requiring developers to change their tools in the development environment*."

These are the practices of how DevSecOps is implemented:

- integrate security tools in the development integration process.

- prioritize security requirements as part of the product's backlog

- collaborate with security and development teams on the threat model

- review infrastructure-related security policies prior to deployment

In addition, applications should follow information security best practices, including issues like data integrity, availability and confidentially, helping developers become aware of how code in a secure way and the need to understand security best practices.

Advantages of implementing DevSecOps

- Early identification of potential vulnerabilities in the code is encouraged

- Greater speed and agility in applying security in all phases of development

- throughout the development process, tools and mechanism are provided to quickly and efficiently respond to changes and new requirements

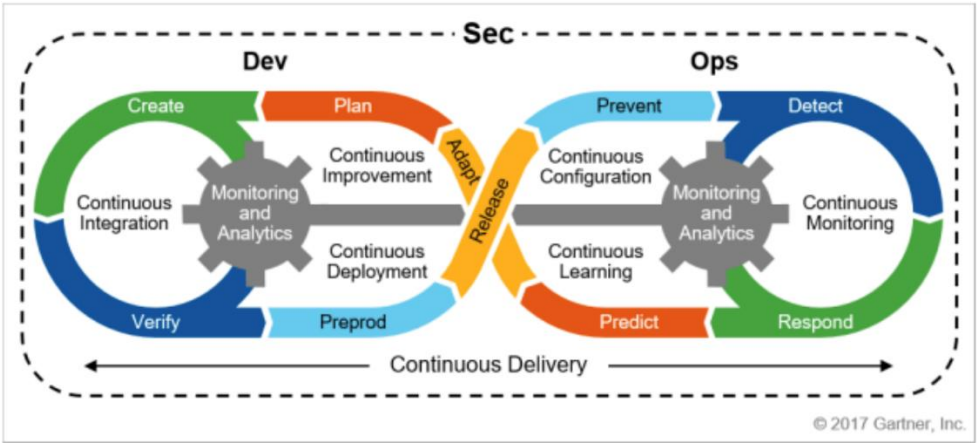
- better collaboration and communications between teams involved in development as in DevOps.

DevSecOps lifecycle

Just like DevOps, DevSecOps suggest the integration of security tools as part of the continuous integration and continuous deployment process.

The integration of these tools makeup pipeline known as ***application security pipeline***.

These pipelines may include phases like *code review automation*, *security testing*, *security scans*, *monitoring* and *automated report generation*.



Applying the DevSecOps methodology

There are six important components in the DevSecOps methodology:

1- **Code analysis**: Deliver the code incrementally, with the aim of being able to detect vulnerabilities quickly.

2- **Change management**: Increase speed and efficiency by allowing changes to come from any source, and then determine if these changes are beneficial through a review process

3- **Compliance monitoring**: Be ready for an audit at any time (RGPD).

4- **Threat research**: Identify potential emerging threats with every code update and respond quickly.

5- **Vulnerability analysis**: identify new vulnerabilities with code analysis, pentesting and architecture analysis, then analyze the respond and patching times.

Security testing

Security testing is often called intrusion testing or penetration testing. This testing can be carried out in two modes:

- **White box testing** allows static analysis, checking the internal functioning of the applications, and having all necessary knowledge through source code and architecture.

- **Black box testing** focuses on examining the functionality of the application without knowledge of the internal structure using dynamic analysis. The test cases of approach focus on exploiting the

interaction with the interaction with application from the outside (APIs, databases, files, protocols, input data and so on) to break the applications security measures.

Security code review

Security code review is an activity that consist of analyzing the software's source code to find out errors and security problems.

This activity can carried out in both traditional and agile development processes.

DevSecOps tools

When implementing DevSecOps, it is important to emphasize the principles and values rather than the use of tools and the people in charge of product must understand the risks and vulnerabilities to which they are exposed if measure are not taken to avoid them.

We divided the tools into several categories that will help us with the different DevSecOps tasks and processes:

* Static Analysis Security Testing (SAST)

* Dynamic Analysis Security Testing (DAST)

* Infrastructure as code security

* Secrets management

* Vulnerabilities management

* Vulnerabilities assessment

Chapter two

Docker container security and best practices

From security point of view, docker containers use the resources of host machine but have their own runtime environment.

This means a container cannot access other containers or the underlying operating system (except the storage volumes to which you gave access) and it will communicate with other networks and containers with the specific network configuration that you want to grant.

Two important concepts

Namespaces: provide isolation for processes and mount points, so processes that run in a container cannot interact with or see processes that run in another container. The isolation of mounting points implies that they cannot interact with the mount points in another container.

Control groups (Cgroups): are a feature of the linux kernel that facilities the limitation of the use of resources at the level of CPU and memory that a container can use. Its ensures that each container gets only the resources it really needs.

Tip: latest version of docker engine support technologies like AppArmor, SElinux, Secure Computing mode (Seccomp).

From security point of view, we must keep in mind docker requires root privileges for working in normal conditions. Docker daemon in responsible for creating and managing containers, which includes creating filesystem, assigning ip addresses, routing packets, process managements and tasks that require administrator privileges.

Tip: it is recommended to ensure that only trusted clients have access if you want to expose the docker daemon outside of your network and use the remote APIs. A simple way is SSL and certification using HTTPs.

Security best practice

The following list summarize the best practice when executing docker containers:

- it is advisable to run the daemon docker process on a dedicated server isolated from other virtual machine.

- special care must be taken to link certain docker host directories as volumes because a container can gain full read and write access and perform critical operations on these directories.

- from point of view of security in communications, the best option is to use SSL-based authentication.

-avoid running processes with root privileges inside the containers.

- we can study the option of enabling specific security profiles, such as AppArmor and SElinux on the docker host.

- all containers share the docker host Kernel, so it is important to have the kernel updated with last security patches.

- one application per container using micro-service-oriented approach

- DO NOT run containers as root, and disable SETUID permissions.

- Use the -cap-drop and -cap-add flags to remove and add capabilities in the container.

- it is advisable not to use environment variables or run containers in privileged mode if you are going to share secrets.

- you must have docker updated to latest version to ensure that all security issues have been solved and also to provide the latest features that docker is incorporating in the core.

- kernel is one the most vulnerable components in container managements as it shared among all containers, so special care should be taken the linux kernel with the latest update.

Tip: from security point of view, it is important to configure the namespaces to limit access to the containers at this point.

Tip: while the container engine must be run with the root user, it is NOT a good practice for containers to do so, and it is necessary to create user for each running container.

- one of the other recommendation for linux systems administrations include the application of the principles of minimum privileges. For this, flags like **read-only** can be applied when executing a container. Limiting the use of filesystem can prevent a potential attacker writing and executing scripts inside the container. The main disadvantage of using the read-only option is that most applications need write files in the directories like /tmp and will not work in the read-only environment. In these case we can use folders and files in which the applications needs writes access and use volumes to mount only those files.

- disabling the SETUID and SETGID permissions.

- verifying image by docker content trust

docker capabilities

docker capabilities allow us to manage the permissions that a process can use to access the kernel and segregate root user privileges to limit actions that can be accessed with privileges.

It is also important to note that container does not have same privileges as the root user of docker host even if it is run as root. This is because docker container run with limited capabilities by default. These include the following

- CAP_SYSLOG: for modifying the behavior of the kernel log
- CAP_NET_ADMIN: for modifying the network configuration
- CAP_SYS_MODULE: for managing kernel modules
- CAP_SYS_RAWIO: for modifying the kernel memory
- CAP_SYS_NICE: for modifying the priority of the processes
- CAP_SYS_TIME: for modifying the system clock
- CAP_SYS_TTY_CONFIG: for configuring tty devices
- CAP_AUDIT_CONTROL; for configuring the audit subsystem

LISTING all capabilities :

Capabilities in linux use libcap package

- getcap: allows listing all the capabilities of a file
- setcap: allows assigning and deleting capabilities of a file
- getpcaps: allows listing the capabilities of a process
- capsh: provide a command line interface for testing and exploring capabilities

disabling ping command in a container

we can use following command that disable the NET_RAW capabilities in the for example python container for disabling ping in a container :

$ docker run -it –cap-drop NET_RAW python sh

execution of privileged containers

sometimes you need your container to have special kernel capabilities that would normally be denied. This can including modifying network settings.

Following command can be used to change the container for example in ubuntu container with full privileged:

$ docker run -ti - - rm - - privileged=true ubuntu sh

Chapter three

Docker host security and best practices

Analyzing the security of docker host is important since most attacks take advantages of kernel vulnerabilities or occur because some package has not been updated. At this point , we will review some tools for auditing the security of the docker host.

Docker daemon security

The most important element of the docker architecture is the docker daemon process that guarantees communication between containers, and the traffic is protected by HTTPs protocol.

Docker works primarily as a client that communicates with a daemon process called ***dockerd*.** This process with root privileges is a socket located in the path ***/var/run/docker.sock***. at this point it is important to note that docker socket exposure can result in privileges escalation.

You must check the access permissions by the users when using the ***/var/run/docker.sock*** socket. In particular , only the root user has to write permissions, and the docker group does not contain users who can compromise the container.

The docker daemon runs with root permissions, so it is important to limit users who have control over the docker daemon. We can give series of recommendation on how we should configure access to the directories and files to the docker daemon.

Auditing files and directories

The docker daemon runs with root privileges, so all directories and files should be constantly audited to know all activities and operations that that are running.

We can use the linux audit daemon framework to audit all events that take place on the host docker.

Auditd It has the following features:

- audit processes and file modification
- monitor system calls
- detecting intrusion
- register commands by users

The linux audit daemon is a framework that allows auditing events on linux systems and is configured using two files, one for the daemon itself (**auditd.conf**) and one for rules used by the ***auditctl*** tool (**audit.rules**).

auditd.conf

this file configures the linux audit daemon (auditd) and focuses on where and how events should be traced. It also defines how to behave when the disk is full, the rotation of log file, and the number of logs to keep . the default settings will be appropriate for most systems.

audit.rules

This file configures which events should be audited.

For example, we can monitor the file located in the path ***/etc/passwd***. We use the following command to indicate to the audit framework which directory or file we want to observe using the path option:

$ auditctl -a exit,always -F path=/etc/passwd -perm=wa

we have to add new rules in the */etc/audit/rules.d/audit.rules* file to correctly configure the audit daemon. Next we will add the necessary rules to be able to audit the directories:

-w /usr/bin/docker -k docker

-w /var/lib/docker -k docker

-w /etc/docker -k docker

-w /usr/lib/system/system/docker.service -k docker

-w /usr/lib/system/system/docker.socket -k docker

-w /etc/default/docker -k docker

-w /etc/docker/daemon.js -k docker

-w /usr/bin/docker-containered -k docker

-w /usr/bin/docker.runc -k docker

After that, we need restart the audit daemon using the following command once the rules have been added:

$ sudo service auditd restart

The logs generated during the audit can be found in the path */var/log/audit/audit.log* if you want to review them.

Kernel linux security and SElinux

Security-Enhanced Linux (SElinux) is a linux kernel security module that provide different security controls like access control, integrity controls, and **Role-Based Access Control** (RBAC).

In addition, it provides privacy policies between docker host and containerized applications.

These tools are **Mandatory Access Control** (MAC) tools that impose security rules in linux to ensure that apart from normal read-write-execute rules that apply to files and processes, more rules can be applied to them at the kernel level.

For example, a MySql process can only afford to write files under specific directories, such as */var/lib/mysql*

Run container with Seccomp profile

Each of the processes that we execute on the operating system have the of interacting with the kernel through system calls. The processes can ask kernel to perform some task, such as modifying file, creating new process, changing the permissions to a directory, or using ***Application Programming Interface*** (APIs), by which the kernel gives access to its services.

Many system calls are accessible to every process in the user area, but a large part are not used for entire life of process. At this point, Seccomp is a tool that allow you limit the exposure of the kernel to system calls by an application. Combined with other tools tat the system offer us, like capabilities and namespaces among others, we have a set designed to secured applications.

Seccomp is a sandboxing facility in the kernel that acts like a firewall for system calls (syscalls). It uses Berkely Packet Filter (BPF) rules to filter syscalls and control how they are handled.

These filters can significantly limit container access to the docker host's linux kernel , especially for simple containers/applications.

Docker uses seccomp in filter mode and has its own JSON-based DSL, that allows you to define profiles that compile down seccomp filters. A container gets the default seccomp profile when you run it, unless you override this by passing *the - -security-opt* flag to the *docker run* command.

For example we are going to block chmod and chown syscalls:

$ *touch profile_policy.json*

And then

{

"defaultAction": "SCMP_ACT_ALLOW",

"syscalls": [

{

"name": "chmod",

"action": "SCMP_ACT_ERRNO"

},

{

"name": "chown",

"action": "SCMP_ACT_ERRNO"

}

]

}

And finally :

$ docker run - - rm -it - -security-opt seccomp:profile_policy.json ubuntu sh

Deny all syscalls

Docker seccomp profiles operate using a whitelist approach that specifies allowed syscalls. Only syscalls on the whitelist are permitted. An empty whitelist, means deny all syscalls.

Docker bench security

Docker bench security is a useful tool to test the security of your docker containers. The objective is to perform the docker CIS checks against a container, and a report is generated than tells you if that container is potentially insecure at the level of permissions and access to the resources.

The tool will inspect the following components:

- Host configurations
- The daemon Docker configuration files
- Image container and compilation files
- Runtime container
- Docker security operations

Auditing Docker host with Lynis

Lynis is an open-source security audit tool for evaluating the security of linux and UNIX-based systems. Lynis executes directly on the docker host so that it has access to the linux kernel.

Once installed, the audit system command performs the following checks:

- Check the operating system
- Perform a search for available tools and utilities
- Check for any Lynis update
- Perform tests with the enabled add-ons
- Perform safety tests by category
- Security scanning status report

For example below command checks the configuration and security of your docker host:

$ lynis audit system

Chapter four

Auditing and Analyzing Vulnerabilities in docker containers

From security point of view, it is important to have knowledge about docker container threats and system attacks, which can impact docker applications. These threats and attacks also applicable to specific docker container versions of the application.

Docker containers threats and attacks

Nowadays it is critical to ensure that the images you are running are up-to-date and do not contain software version with known vulnerabilities. Here are some of the common attacks and threats that containers might suffer:

- **Direct attacks on the kernel** taking advantages of vulnerabilities that has not been patched.
- **Denial of service attacks (DoS):** the main problem is that the container may monopolize the access to certain resources, such as CPU and memory, resulting denial of service.
- **Use of trojanized image**: if an attacker gets someone to execute a trojanized image with malicious code, both the docker host and the data exposed by it are at risk.

Two critical CVE

- **Dirty Cow exploit (CVE-2016-5195):** is a privilege escalation vulnerability in the linux kernel, and it allows any existing user without privileges to perform an elevation of administration privileges.
- **Jack in the box (CVE-2018-8115):** this is a remote code execution vulnerability that affect docker for windows.

Most Vulnerable packages

The following table shows 10 packages that contain most of vulnerabilities in the images:

1. glibc
2. util-linux
3. shadow
4. perl
5. apt
6. openssl
7. tar

8. openldap
9. krb5
10. audit

Analyzing vulnerabilities in docker images

An audit process ensures that all containers are based on updated containers and both hosts and containers are configured securely. Here some of the main features we can validation an audit process:

- **Isolation and minimum privileges:** the containers are executed with the minimum resources and privileges for their execution. For this, it is important to limit both the memory and use of CPU and network functions.
- **Limiting memory and CPU:** limiting the amount of memory available to the containers will prevent attackers from consuming all the memory on the host and killing other services. Also limiting CPU and the network can prevent attackers from executing denial of service attacks.
- **Access controls:** linux security modules, such as AppArmor or SElinux, can be used to enforce access controls and limit system calls.

Specific consideration in an audit process:

- Checking that images and packages are updated with latest version.
- Using base file systems in read-only mode will make it easy to find problems
- Our images should take up as little space as possible. The larger the images, the more difficult the audit will be.
- The kernel of the machine where the docker server is running should always be updated since it is shared point between all containers running on the same server.

Tip: the NVD database, which is managed by U.S government, contain latest vulnerabilities discovered related to the docker ecosystem.

NVD assigns a score from 0 to 10 to each vulnerability. Score of 7-10 are graded as highly critical vulnerability, 4-6 moderate and 0-4 are low vulnerability.

Chapter five

Managing Docker Secrets and Networking

This chapter introduce docker secrets and the essential components of docker networking, including how we can communicate with and link docker containers. We will also review other concepts like port mapping.

With docker uses for exposing the TCP ports that provide services from the container to the host so that users accessing the host can access a container's service.

When creating applications with containers connected to each other, we must use docker networks to be able to communicate with the containers.

Introducing container secrets

Secrets management enable organization to consistently enforce security policies within applications and ensures that only authenticated and authorized entities can access resources in applications, platforms and cloud environments.

The following steps are typically included in a secrets management initiative, many of these approaches and techniques are also used to protect access for those users who have the assigned privileges:

- Authenticate all access request that use credentials and authentication tokens
- Implement the principle of the least privileges
- Enforce Role-Based Access Control (RBAC) and regularly rotate secrets and credentials
- Automate the management of secrets and apply access policies to them
- Remove code secrets, configuration files, and other unprotected areas.

Effective secrets management enables organizations to remove these secrets that we can sometimes see in configuration files and are used CI/CD tools, while offering full audit trials, policy-based RBAC, and secrets rotation.

Docker containers need to access security sensitive data, such as usernames and passwords, SSL certificates, SSH private keys, or any other access restricted information.

In docker, some of this data is provided through environment variables when launching the containers. This is not good practice because when making a list of the process with their invocation parameters, those related to docker will show this information, which is a possible security issue.

What is a secret?

A secret is a piece of information required for authentication, authorization, and encryption. You can use secrets to manage any sensitive information that a container needs at runtime, but you don't want to store it directly in the image.

With docker secrets, you can manage this information that is needed at runtime but does not want to be stored in the docker image or in the source code repository. Some examples of sensitive information are as follows:

- Usernames and Passwords
- TLS certifications and keys
- SSH keys
- The name of databases or internal server

Another use case for using secrets is to provide an abstraction layer between the container and set of credentials. Consider a scenario where you have separate development, test, and production environments for your application.

Each of these environments can have different credentials, stored in the development, test, and production environments with the same secret name. this way containers only need to know the name of the secret for working in all three environments.

Managing secrets in Docker

Docker secrets are provided to the containers that need them and transmitted in encrypted form to the node on which they run, secrets are mounted on the filesystem at the ***/var/secrets/<secret-name>*** path in a decrypted way and can be accessed by the container service.

Until now, creating a service on swarm means having its configuration within the images, available on all hosts locally or mounted via network storage. That said, secrets can contain files, so we can use them to easily manage the configuration of the services since the information will be available on all the hosts that execute some service task.

It is common that in an image we need to use credentials and access tokens or files with information that we don not want to share. If we pass these elements to the image using commands like ***copy*** or ***add***, they will be visible in the image and anyone who has access ti it will be able to see them.

So, it is important to mention that docker secrets are only available to swarm services, not to standalone containers. That means the secrets can be pushed to containers on when containers are running as swarm as a swarm service.

Chapter six

Kubernetes Security

In this chapter, we will learn about Kubernetes security and best practices for securing components and pods by applying the principles of least privileges in Kubernetes.

Kubernetes characteristic

From perspective of DevOps, Kubernetes has the following characteristic:

- **Operating in the DevOps model:** in the DevOps model, software developers assume greater responsibility for building deploying applications.
- **Creation of common service sets:** Applications request a service from another application pointing to an ip address and port number. With Kubernetes we can build applications in containers that provide services that are available for other containers to use.
- **DataCenter, Pre-Configuration:** Kubernetes aims to create consistent APIs that result in stable environment for running applications in containers. Developers should be able to create applications that work in any cloud provider that supports those APIs. This reliable framework means developers can identify the version of Kubernetes, along with the services they need, and not have worry about the specific configuration of the data center.

Configuring Kubernetes

While docker manages entities referred to as image and containers, Kubernetes wraps those entities in what is referred to as pods. A pod can contain one or more running containers and is the unit that manages Kubernetes. Kubernetes brings several advantages to container management as pods:

- **Multiple nodes:** instead of simply deploying a container on a single host, Kubernetes can implement a set of pods on multiple nodes. Essentially, *a node provides the environment where a container is executed.*
- **Replication:** Kubernetes can act as a replication controller for a pod. This means you can set how many replicas of a specific pod should be running at all times.
- **Services:** The word "service" in the context of Kubernetes implies that you can assign a service name (ID) to a specific ip address and port and then assign a pod to provide that service. Kubernetes internally tracks the location of that service and can redirect requests from another pod of that service to the correct address and port.

You must understand the following concepts if you choose to configure Kubernetes:

- **Kubernetes Controller:** A Kubernetes controller acts as a node from which the pods, replications controller, services, and other components of a Kubernetes environment are implemented and managed. You must configure and run the *systemd, kube-api-server, kube-controller-manager and kube-scheduler* services to create a Kubernetes controller.
- **Kubernetes node:** A Kubernetes node provides the environment in which the containers run. To run a machine as a Kubernetes node, it must be configured to run the docker, *kube-proxy and kubelet* services. These services must be run on the Kubernetes cluster's each node.

- **Kubectl command:** most Kubernetes administration is performed on the master node using *Kubectl* command. With Kubectl, we can create, obtain, describe, or eliminate any of the resources that Kubernetes manages.
- **Resources files (YAML or JSON):** The *Kubectl* command expects us the information needed to create that resource to be in one of these two types formats when you create a pod, a replication controller, services, or another resource in Kubernetes.

The classical way to see how Kubernetes works is to configure a Kubernetes cluster that has a master controller node and at least two nodes, each operation on separate system. The latest method of setting up a highly available Kubernetes cluster allow splitting up the master component onto multiple nodes of Orchestration/Control Plane and ETCD.

The Kubernetes API, managed by a kublet, must be protected to ensure that it is not accessed in an unauthorized way to perform malicious actions. If unauthorized access was made to one of the containers running in a pod of a Kubernetes environment, the API can be attacked by means of some simple commands to be able to visualize the information about the entire environment.

Security in Kubernetes should be focused on preventing image manipulation and unauthorized access to the entire environment. Regarding runtimes protection, it is essential not to deploy pods with root permissions, checking that pods have defined security policies and that Kubernetes is using secret for credential and password management.

For example, attackers can execute remote code execution attacks that can give them access to the cluster anonymously if we have a misconfigured kubelet. The kubelet maintains a set of pods within a Kubernetes cluster and functions as a local agent that monitors the pod specifications through the Kubernetes API server.

Kubernetes security best practices

It is advisable to follow some best practices at the security level due to the impact that some implementations that can be carried out in an organization can cause.

Using Secrets

If we want to start securing our Kubernetes projects, we can start with good practices like not storing objects with sensitive data like passwords, SSH keys, or Auth tokens in the clear. The use of secrets allows you to control how sensitive data is used, and it significantly reduces the risk of exposure of that sensitive data to unauthorized users.

Firewall ports

This security practice is frequently used since it is not advisable to expose a port that does not need to be exposed. It is best to define the port's exposure to prevent this from happening.

The first thing you should do is check the existence of some interface or define an IP to link the service; for example, the localhost interface 127.0.0.1. Some processes are opening so many ports on all interfaces that they should rather have a public access firewall. Although they only allow purely confidential information, they also allow you direct access to your set of computers.

Restrict the Docker pull <image> command

Docker is a resource that can sometimes be uncontrolled by the ease of access it has. That is, anyone with access to the Kubernetes API or Docker connector can obtain the image they want, generating traffic from infected images or serious security problems for Kubernetes. Many clusters have also become a network of Bitcoin miners.

Although it is a problem that seems not to be solved, the Image Policy plugin can significantly improve that situation, connecting directly with the Docker API. This plugin imposes a series of strict security rules that reflect a black and white list of images that can be extracted.

Another solution is to use the Image Policy Webhook through Admission Controller, which intercepts all image extractions and takes care of security just like the plugin mentioned earlier.

API authorization and anonymous authentication

You should know what authorization mode your system is using. This can be done by verifying the parameters, where you can also check if authentication is configured anonymously.

It is important to know that this configuration will not affect the kubelet authorization mode since it exposes an API on its own that executes commands that kubelet can completely ignore.

More specifically, a kubelet provides a command API used by kubi-apiserver, in which arbitrary commands are executed on a specific node.

This configuration can be designed as - -authorization-mode = Webhook

and - -anonymous -auth = false.

When we talk about giving permissions in a Kubernetes cluster, we will have to talk about Role Based Access Control or RBAC, which manages security policies for users, groups or Pods. It is implemented in a stable way in the latest versions of Kubernetes. You can use Roles and ClusterRoles to define access profiles.

We are defining specific rules for accessing our pods in the following example:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole metadata:

name: cluster-role rules:

-apiGroups: run resources: ["pods"1 verbs: ["get", "list"]

You can use open source tools like rbac-manager https://github.com/FairwindsOps/rbac-manager to help you simplify the authorization process in Kubernetes using RBAC to facilitate RBAC configuration.

This is an operator that supports declarative configuration for RBAC with new custom resources. Instead of managing role bindings or service accounts directly, you can specify a desired state and the RBAC manager will make the necessary changes to achieve that state.

Management of resources and limits

It is important to manage the resources and limits that we are going to assign to our applications when creating a container in a Kubernetes infrastructure, especially in production. At the security level, it is important because a single container can generate a denial of service when sharing a host with other containers. In the generation of the Pod, we can easily control it through the requests and limits sections in the deployment execution file.

Security features built into K8s

Kubernetes offers native security features to protect against some of the threats described earlier or at least mitigate the potential impact of a breach.

The main safety features include:

• **Role-Based Access Control (RBAC):** Kubernetes allows administrators to define what are called Roles and ClusterRoles that specify which users can access which resources within a namespace or an entire cluster. This way, RBAC provides a way to regulate access to resources.

• **Pod security policies and network policies:** Administrators can configure pod security policies and network policies, which place restrictions on how containers and pods can behave. For example, pod security policies can be used to prevent containers from running as root users, and network policies can restrict communication between pods.

• **Network encryption:** Kubernetes uses TLS encryption by default, which provides additional protection for encryption of network traffic.

These built-in Kubernetes security features provide layers of defense against certain types of attacks, but they do not cover all threats. Kubernetes does not offer native protections against the following types of attacks:

• **Malicious code or incorrect settings inside containers or container images:** A third-party container scanning tool must be used to scan them.

• **Security vulnerabilities in host operating systems:** Again, these need to be searched with other tools. Some Kubernetes distributions like OpenShift integrate security solutions like SELinux at the kernel level to provide more security at the host level, but this is not a feature of Kubernetes itself.

• **Container runtime vulnerabilities:** In this case, Kubernetes has no way of alerting if a vulnerability exists within its runtime or if an attacker is trying to exploit a vulnerability at the time of execution.

• **Kubernetes API abuse:** Kubernetes does nothing to detect or respond to API abuse beyond following any RBAC and security policy settings that you define.

• **Management tools vulnerabilities or configuration errors:** Kubernetes cannot guarantee that management tools like Kubectl are free from security issues.

Managing secrets

A secret is everything that nobody else in the cluster should know, neither the rest of the applications nor users that access the cluster. For example, a password from a certificate store, an API key so that an application can consume third-party resources, and so on.

Let's say that someone discharges those resources along with certain permissions. From there, it is the application that requests those secrets from K8s by presenting the information that authorizes them to consume those resources.

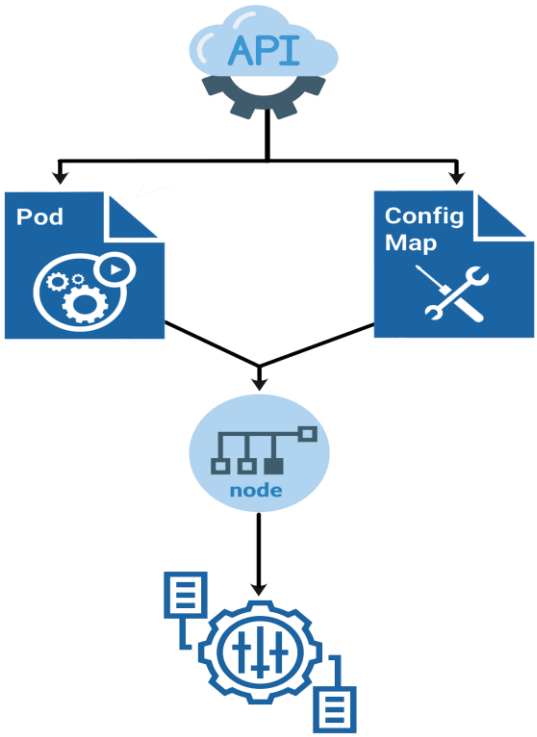
Authorization management is done through what is known as Role-Based Access Control (RBAC), that is, the application can access certain types of resources only if it has a certain role. Additionally, it's important to configure these roles and release the secret before the application is deployed.

Kubernetes secrets

Using secrets allows you to control how sensitive data is used and significantly reduces the risk of exposure of sensitive data to unauthorized users. This information is often placed in pod specifications or container images. A secret can be generated both by a user and by the system itself.

When the system does this, secrets are automatically generated by service accounts with API credentials. Kubernetes automatically creates secrets that contain credentials to access the API and modifies your Pods to use this type of secret.

The following image illustrates a basic diagram for storing secrets in the cluster:



Other interesting facts about secrets:

• Secrets are objects with namespaces, that is, they exist in the context of a namespace

• You can access them through a volume or environment variable from a container running in a pod.

Handle security risks in Kubernetes

Here are the main strategies that we can follow to manage the risks of putting your application with Kubernetes in production:

• **Integrate security from the early stages of development:** With Kubernetes, it is necessary to integrate security at each stage of the software development process. It is a mistake to leave security settings for the last step as it may be too late.

• **Consider a commercial platform of Kubernetes:** When you participate in a Kubernetes trading platform, the most important benefit you get is the rapid structural responses from development to any threat or problem. Kubernetes will be updated quickly to any vulnerability, and you will always have the latest security updates for your company.

• **Do not trust your old tools and practices:** The attackers update faster than the software, so the same moves at any time may be obsolete. You should not assume that your conventional security tools will protect you. Many open source tools evaluate Kubernetes clusters or perform penetration tests on clusters and nodes. Experts point out that it is necessary to keep your software updated and patched, for example, and new approaches and tools are also necessary.

Here is a summary of the key parts of a Kubernetes environment and the most common security risks that affect them:

• **Containers:** Containers can contain malicious code that was included in your container images. They can also be subject to misconfigurations that allow attackers to gain unauthorized access under certain conditions.

• **Host operating systems:** Vulnerabilities or malicious code within operating systems installed on Kubernetes nodes can provide attackers with a path to Kubernetes clusters.

• **Container runtimes:** Kubernetes supports a variety of container runtimes. All of them can contain vulnerabilities that allow attackers to take control of individual containers, escalate attacks from container to container, and even gain control of the Kubernetes environment.

• **Network layer:** Kubernetes relies on internal networks to facilitate communication between nodes, pods, and containers. It also often exposes applications to public networks so that they can be accessed over the Internet. Both network layers can allow attackers to gain access to the cluster or escalate attacks from one part of the cluster to others.

• **Kubectl Dashboard and other management tools:** They may be subject to vulnerabilities that allow abuse in a Kubernetes cluster.

Analyzing Kubernetes components security

Pods are the main component of Kubernetes and represent one or more containers that share network and storage configurations. So, their security is very important and needs to be implemented from the first steps of its design, using security policies.

The official documentation provides some examples of how to apply these security policies in our implementation with Kubernetes. It is available at https://kubernetes.io/docs/concepts/policy/pod-security-policy.

According to official documentation, a pod security policy is a cluster-level resource that controls aspects of a pod's security. These security policies are defined through the PodSecurityPolicy object, through which we can define the conditions which a pod must meet to be accepted in the system. It also allows us to define the default values of fields that are not explicitly assigned.

A security policy is defined as practically everything in Kubernetes, through a manifest file, usually in YAML format. Let's consider an example:

apiVersion: policy/vlbeta1 kind: PodSecurityPolicy

metadata:

name: permissive spec:

privileged: true

hostNetwork: true

hostIPC: true

hostPID: true seLinux:

rule: RunAsAny

supplementalGroups:

rule: RunAsAny

runAsUser:

rule: RunAsAny

fsGroup:

rule: RunAsAny

hostPorts:

- min: 0

max: 65535 volumes:

In this example, you can see how the defined policy is very permissive. It practically allows us to run a pod with all kinds of privileges. For example, we can execute it in privileged mode (privileged: true) so that we can have access to parts of the host; share the space of network names, processes, and Inter-Process Communication (IC) of the host; run the container or containers as root; etc. Such configurations should be avoided unless there is a good reason.

Pod security policies

Pod security policies allow administrators to control the following aspects:

• **Containers in privileged mode:** This feature allows or does not allow the execution of containers in privileged mode. The field that sets this aspect is called privileged. The containers run in non-privileged mode by default. Here are some of the main values that this feature can take:

• **Host namespace:** There are four fields that allow us to define the behavior of a container with respect to access to certain parts of the host:

• HostPID: This controls whether the pod containers share the same process space (IDs) of the host.

• **HostIPC:** This controls whether the containers in a pod share the host's IPC space.

• **HostNetwork:** This controls whether a pod can use the same host network space. It implies that the pod would have access to the loopback device and the processes running on that host.

• **HostPorts:** This defines the range of ports allowed in the host network space. This range is given by the HostPortRange field, and the min and max attributes that define the range of ports are included in the range.

• **Volumes and filesystems:** Here are some of the main values that this feature can take:

• **Volumes:** This provides a list of permitted volumes, and they correspond to the source used to create the volume.

• **FSGroup:** Allows you to indicate the groups where to apply certain volumes.

• **AllowedHostPaths**: Specifies a list of paths allowed to be used by volumes. An empty list would imply that there are no restrictions.

This list is defined by two attributes: pathPrefix and readonly.

• **ReadOnlyRootFilesystem:** This requires that the containers run with the root filesystem in read-only mode.

• **Users and groups:** Some of the main values of this feature are:

• **RunAsUser:** Specifies which user the containers run inside the pod.

• **RunAsGroup:** Specifies with which group ID the containers run within the pod

• **Privilege escalation:** Basically, it controls the no_new_privs option of the container process. This option prevents binaries with the setuid option from changing the user's effective ID and prevents enabling new extra capabilities. Here are some of the main values that this feature can take:

• **allowPrivilegeEscalation:** Specifies whether or not to set the security context of the container.

By default, allowPrivilegeEscalation = true to avoid problems with binaries with setuid active.

• **DefaultAllowPrivilegeEscalation:** This allows you to set the default option of allowPrivilegeEscalation.

• **Capabilities:** GNU/Linux capabilities are a series of superuser privileges that can be enabled or disabled independently. The following fields accept the capabilities as a list, without the CAP.prefix (all capabilities in GNU/Linux begin with that prefix):

• **AllowedCapabilities:** List of capacities that can be added to a container. All capacities are allowed by default. If this field is specified empty, it implies that you cannot add capacities to a container beyond those defined by default. The asterisk (*) can be used to refer to all capabilities.

• **RequiredDropCapabilities:** List of capacities that must be removed from the container. These are removed from the default capacity group. The capabilities included in this field should not be included in AllowedCapabilities or DefaultAddCapabilities.

• **DefaultAddCapabilities:** Capabilities added to a default container by default.

Static analysis with kube-score

kube-score is a tool that performs static code analysis of your Kubernetes object definitions. The output is a list of recommendations of what you can improve to make your application more secure and resilient.

Auditing the state of the cluster

You may have to perform a small internal audit of the state of the cluster when you work with Kubernetes clusters. We can use the Polaris tool available in the GitHub repository at https://github.com/FairwindsOps/polaris to do this.

This tool can be used in three ways:

• In audit mode, where it shows us the state of the cluster and whether there is any aspect that we can improve.

• In validation mode, that allows us to validate what we are going to execute complies with the standard.

• In YAML file testing mode via command console, where it allows us to check our developments locally.

Using livenessProbe and readinessProbe

Health checks are very important in Kubernetes. Two types of controls are provided at this point: livenessProbe and readinessProbe:

• *livenessProbe* is used to check if the application is still running or has stopped. Kubernetes does nothing if the application runs successfully, but it will launch a new pod and run the application in it if your application is stopped.

• *readinessProbe* is used to verify that the application is ready to start sending traffic. Kubernetes will stop sending traffic to the pod until this health check fails.

Setting limits and resource requests

The application will stop working when you are deploying a large application on a resource-constrained production cluster where nodes run out of memory or CPU. This application dowauche can have a huge impact on your business, but you can solve this by having requests and resource limits.

Requests and resource limits are the Kubernetes mechanisms for controlling the use of resources like memory and CPU. If one pod consumes all CPU and memory, the other pods will run out of resources and be unable to run the application.

We can set requests and limits for each container in a pod to improve this aspect. CPU is defined using millicores and memory using bytes (megabyte / mebibyte). In the following example, we are setting a CPU limit of 500 millicores and 128 mebibytes, and we are setting a quota for CPU requests of 300 millicores and 64 mebibytes:

containers:

- name: prodcontainer1

image: ubuntu resources:

requests:

memory:

"64Mi"

cpu:

#300m"

limits:

memory: "128Mi" cpu:

1500m"

Applying affinity rules between nodes and pods

One of the main mechanisms in Kubernetes for associating a pod with a node within the cluster is to define the affinity for better performance. We can use node affinity to define the criteria that a pod will follow to associate with a certain node in a Kubernetes cluster:

apiVersion: V1 kind: Pod metadata:

name: ubuntu spec:

affinity:

nodeAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 2 preference:

matchExpressions:

- key: disktype operator: In values:

- ssd

containers:

- name: ubuntu image: ubuntu

imagePullPolicy: IfNotPresent

We can use pod affinity to schedule multiple pods on the same node (to improve latency) or decide to keep pods on separate nodes (for high availability) to increase performance.

apiversion: V1 kind: Pod metadata:

name: ubuntu-pod spec:

affinity: podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: security operator: In values:

- S1 topologykey: failure-domain.beta.kubernetes.io/zone

containers:

- name: ubuntu-pod

image: ubuntu

After analyzing the cluster workload, we will have to decide on the best affinity strategy to use.

Chapter seven

Auditing and Analyzing Vulnerabilities in Kubernetes

KubeBench security

KubeBench (https://github.com/aquasecurity/kube-bench) is a Kubernetes security scanner that allows us to eliminate about 95% of configuration defects, generating specific guidelines to ensure the configuration of your computer network through the application of Kubernetes benchmark.

CIS benchmarks for Kubernetes with KubeBench

We can use KubeBench to verify the rules of CIS Benchmark.. It is a tool that will automate the entire process of validating CIS Benchmark rules for Kubernetes. We can install KubeBench through this dedicated container by executing the following container:

https://hub.docker.com/r/aquasec/kube-bench

This tool supports tests for multiple versions of Kubernetes defined in the

CIS guides, and the easiest way to run this tool is to run it from a container and launch the tests on the Kubernetes cluster with the following command:

$ docker run - -rm -v 'pwd: /host aquasec/kube-bench: latest install*

Kubernetes security projects

we will review different security projects that can help us, both to secure our Kubernetes cluster and to offer the best possible performance to our infrastructure.

Kube-hunter

Kubernetes clusters are mounted on a set of nodes or servers in which at least one has to take the role of master. The rest are defined as workers and have visibility with each other in order to communicate.

is tool relies on known attack vectors and information about the attack analysis in a Kubernetes installation. Place of its environments installations you to perform a security vulnerability.

Mallows remote, internal, or CID scanning over a Kubernetes cluster and incorporates an active option through which it tries to exploit the findings. It can be run locally or through the deployment of a container that is already prepared.

we can run this tool in several ways: locally from the source code, using a container, or using a pod. In the case of the basic installation from source code, we have to install a series of dependencies, clone the GitHub repository, and run the kube-hunter script. The commands to execute in this

case are:

$ git clone https: //github.com/aquasecurity/kube-hunter.git

$ cd ./kube-hunter && pip install -r requirements.txt

$ ./kube-hunter. py

We can use the following command in the case of using a Docker container:

$ docker run -rm aquasec/kube-hunter

We can use the -cidr parameter to specify a network to scan, as shown here:

$ docker run -rm aquasec/kube-hunter -cidr 192.168.0.0/24

Regarding the scan options, kube-hunter will open an interactive session, where you can select one of the following scan options. For example, you can specify remote machines using the - remote option, as follows:

§ kube-hunter.py -remote domain.com

To control the log, we can specify a log level using the - log option. Consider this example:

$ kube-hunter.py -active - log WARNING

Kubesec

This tool (https://kubesec.io) allows you to analyze the security risk for Kubernetes resources. Here are some of the main features:

• Helps you quantify the risk for Kubernetes resources

• Runs against your Kubernetes applications (deployments and pods)

• Can be used as a standalone application or as kubectl plugin

https://github.com/controlplaneio/kubectl-kubesec

In the following URL, we can execute kubesec over a Kubernetes security scenario:

https://www.katacoda.com/controlplane/scenarios/kube-sec-deploy

Kubectl plugins for managing Kubernetes

There are many plugins for kubect1 to interact with and perform all kinds of operations against our cluster. We have seen that kubect1 is the command-line tool to interact directly with Kubernetes, and it also allows you to create custom plugins, increasing your possibilities by adding ad-hoc commands to the existing ones.

We can review some plugins that offer us different security and control features to make our implementation with Kubernetes much safer. Some plugins are focused, for example, on the security of the pods, and others in RABC, and we will even see one that will allow us to sniff all the network traffic generated to or from a pod.

kubectl-trace

kubect1-trace (https://github.com/iovisor/kubectl-trace) is a plugin that allows using bpftrace in a Kubernetes cluster with the aim of creating control points in the execution to manage its flow, or even stop it, detect problems, and make an in-depth analysis of the infrastructure. You can find the complete bpitrace manual at github.

Kubectl-debug

kubect1-debug (https://github.com/aylei/kubectl-debug) is a plugin that complements perfectly with kubect1-trace for debugging tasks. This allows you to execute a container within a pod that is running. It shares the namespace of the processes (PID, network, user, and IPC of the container to be analyzed, allowing us to debug them without having to install anything beforehand.

You can see a demonstration of its use at https://github.com/aylei/kubectl-debug/blob/master/docs/kube-debug.gif.

Ksniff

There is another plugin called sniff https://github.com/eldadru/ksniff that lets us analyze all the network traffic of a Kubernetes pod using tcpdump and Wireshark.

Ksniff uses the data collected by tcpdump associated with a pod and then sends it to Wireshark to perform the analysis. This plugin is essential if you are working with microservices since it is tremendously useful for identifying errors and problems between them as well as their dependencies.

kubectI-dig

Sometimes, getting the information from a Kubernetes cluster requires the use of several commands, which, in turn, return all kinds of information.

Thanks to this plugin https://github.com/sysdiglabs/kubectl-dig; you can install a user-friendly user interface to easily see all the information related to the Kubernetes cluster.

Rakkess

Access control to all the elements of a Kubernetes cluster is one of the main tasks in securing it. From kubectl, we can obtain this information from a resource, but we cannot get an overview.

Rakkess plugin (https://github.com/corneliusweig/rakkess) allows us to obtain a complete list in a matrix form of the current situation of access permissions between users and all server resources.

Kubestriker

Kubestrikerhttps://github.com/vchinnipilli/kubestriker) is a platform-agnostic tool designed to tackle Kubernetes cluster security issues due to misconfigurations and helps strengthen the overall IT infrastructure of any organization.

It performs numerous in-depth checks on a range of services and open ports well across more than one platform, such as self-hosted Kubernetes, Amazon EKS, Azure AKS, Google GKE, and so on, to identify any misconfigurations that make organizations an easy target for attackers.

In addition, it helps safeguard against potential attacks on Kubernetes clusters by continuously scanning for anomalies. Furthermore, it comprises the ability to see some components of Kubernetes infrastructure and provides visualized attack paths of how hackers can advance their attacks.

There are several ways to install and run this tool. For example, we can run a Docker container with the following commands:

$ docker run -it --rm -v /Users/<yourusername>/.kube/config:/root/.kube/config-v

"$(pwd)":/kubestriker --name kubestriker cloudsecguy/kubestriker:v1.0.0

$ python -m kubestriker