1.a To understand Devops Principle practice and devops engineer role and responsibilities

**Principles**

**Collaboration**

The goal of DevOps principles is to create a culture of communication and collaboration between development and operations teams. This often involves merging the two teams into a single unit that covers the full application lifecycle. DevOps teams are responsible for ensuring that every aspect of the product meets high standards. This means that they take on full responsibility for developing features or products from start to finish. As a result, they need to have a good understanding of all aspects of the product. This increased investment and attachment from the team leads to higher quality output.

**Automation**

In a DevOps culture, automation is seen as essential for achieving speed and reliability. Automation can help to reduce the time it takes to carry out tasks such as building, testing, and deploying code. It can also help to ensure that tasks are carried out consistently and accurately.Automation is central to the CI/CD pipeline, which is a key DevOps practice. The CI/CD pipeline is a set of automated processes that take code from development through to production. It includes steps such as code checking (continuous integration), testing (continuous delivery), and deployments.

**Continuous Improvement**

Continuous improvement is a fundamental principle of agile practices, lean manufacturing, and Improvement Kata. It is also a key component of DevOps principles and culture.In a DevOps culture, continuous improvement means that teams are constantly looking for ways to improve their processes and tools. They also work to Automate tasks wherever possible. This helps to ensure that the team is always learning and improving, so that they can deliver value faster and more reliably. One of the key ways in which DevOps teams achieve continuous improvement is through feedback loops. Feedback loops are a key part of the DevOps philosophy, and involve constantly seeking feedback from all stakeholders in the software development process. This feedback is used to make improvements to the way that software is delivered.

**Customer-centricity**

To create products and services focused on user requirements, DevOps teams utilize short feedback loops with users and customers. This helps to ensure that the products and services delivered are of high quality and meet customer expectations. In a DevOps culture, customer feedback is seen as an essential part of the development process. Feedback should be sought at every stage of the development process, from initial requirements gathering through to final deployment.

**Holistic thinking**

In a DevOps culture, teams work to understand the full context of their work. This includes understanding the dependencies between different parts of the system, and how changes in one area can impact other areas. Holistic thinking helps to ensure that DevOps teams can make informed decisions about the best way to deliver value. It also helps teams to avoid making changes that could have negative impacts elsewhere in the system.

**Roles and Responsibilities**

The main function of a DevOps engineer is to introduce the continuous delivery and continuous integration workflow, which requires the understanding of the mentioned tools and the knowledge of several programming languages. Depending on the organization, job descriptions differ. Smaller businesses look for engineers with broader skillsets and responsibilities. For example, the job description may require product building along with the developers. Larger companies may look for an engineer for a specific stage of the DevOps lifecycle that will work with a certain automation too

The basic and widely-accepted responsibilities of a DevOps engineer are:

* Writing specifications and documentation for the server-side features
* Continuous deployment and continuous integration (CI/CD) management
* Performance assessment and monitoring
* Infrastructure management
* Cloud deployment and management
* Assistance with DevOps culture adoption

1.b To understand devops cycle tools and find out Vulnerabilities and attacks on life cycle tools

**Devops Life Cycle**

DevOps requires a delivery cycle that comprises planning, development, testing, deployment, release, and monitoring with active cooperation between different members of a team.

**Agile planning**

In contrast to traditional approaches of project management, Agile planning organizes work in short iterations (e.g. sprints) to increase the number of releases. This means that the team has only high-level objectives outlined, while making detailed planning for two iterations in advance. This allows for flexibility and pivots once the ideas are tested on an early product increment. Check our Agile infographics to learn more about different methods applied.

**Continuous development**

The concept of continuous “everything” embraces continuous or iterative software development, meaning that all the development work is divided into small portions for better and faster production. Engineers commit code in small chunks multiple times a day for it to be easily tested. Code builds and unit tests are automated as well.

**Continuous automated testing**

A quality assurance team sets committed code testing using automation tools like Selenium, Ranorex, UFT, etc. If bugs and vulnerabilities are revealed, they are sent back to the engineering team. This stage also entails version control to detect integration problems in advance. A Version Control System (VCS) allows developers to record changes in the files and share them with other members of the team, regardless of their location.

**Continuous integration and continuous delivery (CI/CD)**

The code that passes automated tests is integrated in a single, shared repository on a server. Frequent code submissions prevent a so-called “integration hell” when the differences between individual code branches and the mainline code become so drastic over time that integration takes more than actual coding.Continuous delivery, detailed in our dedicated article, is an approach that merges development, testing, and deployment operations into a streamlined process as it heavily relies on automation. This stage enables the automatic delivery of code updates into a production environment.

**Continuous deployment**

At this stage, the code is deployed to run in production on a public server. Code must be deployed in a way that doesn’t affect already functioning features and can be available for a large number of users. Frequent deployment allows for a “fail fast” approach, meaning that the new features are tested and verified early. There are various automated tools that help engineers deploy a product increment. The most popular are Chef, Puppet, Azure Resource Manager, and Google Cloud Deployment Manager.

**Continuous monitoring**

The final stage of the DevOps lifecycle is oriented to the assessment of the whole cycle. The goal of monitoring is detecting the problematic areas of a process and analyzing the feedback from the team and users to report existing inaccuracies and improve the product’s functioning.

**Infrastructure as a code**

Infrastructure as a code (IaC) is an infrastructure management approach that makes continuous delivery and DevOps possible. It entails using scripts to automatically set the deployment environment (networks, virtual machines, etc.) to the needed configuration regardless of its initial state.Without IaC, engineers would have to treat each target environment individually, which becomes a tedious task as you may have many different environments for development, testing, and production use. Having the environment configured as code, youCan test it the way you test the source code itself and Use a virtual machine that behaves like a production environment to test early. Once the need to scale arises, the script can automatically set the needed number of environments to be consistent with each other.

**Containerization**

Virtual machines emulate hardware behavior to share computing resources of a physical machine, which enables running multiple application environments or operating systems (Linux and Windows Server) on a single physical server or distributing an application across multiple physical machines. Containers, on the other hand, are more lightweight and packaged with all runtime components (files, libraries, etc.) but they don’t include whole operating systems, only the minimum required resources. Containers are used within DevOps to instantly deploy applications across various environments and are well combined with the IaC approach described above. A container can be tested as a unit before deployment. Currently, Docker provides the most popular container toolset.

**Microservices**

The microservice architectural approach entails building one application as a set of independent services that communicate with each other, but are configured individually. Building an application this way, you can isolate any arising problems ensuring that a failure in one service doesn’t break the rest of the application functions. With the high rate of deployment, microservices allow for keeping the whole system stable, while fixing the problems in isolation. Learn more about microservices and modernizing legacy monolithic architectures in our article.

**Cloud infrastructure**

Today most organizations use hybrid clouds, a combination of public and private ones. But the shift towards fully public clouds (i.e. managed by an external provider such as AWS or Microsoft Azure) continues. While cloud infrastructure isn’t a must for DevOps adoption, it provides flexibility, toolsets, and scalability to applications. With the recent introduction of serverless architectures on clouds, DevOps-driven teams can dramatically reduce their effort by basically eliminating server-management operations.

**Tools**

Server configuration tools are used to manage and configure servers in DevOps. Puppet is one of the most widely used systems in this category. Chef is a tool for infrastructure as code management that runs both on cloud and hardware servers. One more popular solution is Ansible that automates configuration management, cloud provisioning, and application deployment.

CI/CD stages also require task-specific tools for automation — such as Jenkins that comes with lots of additional plugins to tweak continuous delivery workflow or GitLab CI, a free and open-source CI/CD instrument presented by GitLab.

Containerization and orchestration stages rely on a bunch of dedicated tools to build, configure, and manage containers that allow software products to function across various environments. Docker is the most popular instrument for building self-contained units and packaging code into them. The widely-used container orchestration platforms are commercial OpenShift and open-source Kubernetes.

Monitoring and alerting in DevOps is typically facilitated by Nagios, a powerful tool that presents analytics in visual reports or open-source Prometheus.

**Vulnerabilities and attacks on devops tools**

**Reckless use of recycled code**

DevOps enhances developer productivity considerably by allowing them to reuse code from previous projects omgomg or open source repositories like GitHub. While focusing on productivity, proper screening and sanitization are essential else these codes can pose threats to an application.

**Containers make your applications more vulnerable**

Bringing the benefits of standardization and isolation, containers are spread across different systems. And the microservice approach involving a distributed infrastructure exposes your system to the network making the threat landscape larger and providing additional attack vectors. As containers are highly replicable, it becomes really easy for cyber attackers to sneak into your system.

**Not so safe Hosts**

Are you done by securing the containers? Think again! What about the hosts they are not automatically safe as they are vulnerable to day-zero attacks and ever-evolving new threats. It is essential to provide proper protection to hosts and also ensure container runtime is up-to-date to avoid risks.

**Manage your sensitive data well**

When leveraging DevOps, it is essential to create privileged accounts and login details. A host of useful data including:

**Confidential data**

Keys , Database passwords , Storage account credentials Embedded passwords must be stored in repositories and kept extremely secure as this information can be used by attackers for malicious activities.

Rethink using Multiple platforms

Running multiple workloads on one single platform is not a good idea. Your sensitive data/workloads should be distributed across a dedicated set of machines to avoid cyber attack from neighborhood applications. Further, it is highly important that sensitive metadata should be secured properly.