Devops Tutorial

1. **Explain how devops extends agile principles to the entire systems lifecycle.**

   Systems lifecycle stages in devops

   Plan
   Develop
   Deliver
   Operate

   In the plan phase, devops team ideates, define and describe features and capabilities of application and system. Here they create back logs, track bugs and also manage agile software development with scrum.
   In the development phase, coding, testing, reviewing and integration of code is also done. The product is implemented in small increments through continuous integration and automated testing.
   In the delivering phase, small increments are delivered to the customer with manuals also.
   In the operate phase maintaining, monitoring and troubleshooting of application is done.

   This cycle is done in shorter release cycles so that it makes planning and risk management easy. Because the progress is incremental it reduces impact on system stability.

2. **What are the similarities and differences between centralized and distributed version control systems?**

   Similarities

   They both used to keep track of all the project work done by every other developer in the project.
   They both have a central server repository to communicate (but they work in different ways).

   Differences

Centralized	**Distributed**
Keep the history of changes on a central server from which everyone requests the latest version of the work and pushes the latest changes	Everyone has a local copy of the entire work's history
Slow when it comes to process transactions and working with tags and branches	Considerably faster and easier to work with when it comes to creating and merging branches and tags
Requires internet connection	Work offline and gives flexibility
Slower response – need to communicate with remote server	Faster response
If the project has long history or the project contain large binary files – just need to get few lines of code because you	If the project has long history or the project contain large binary files – downloading the entire project in DVCS

don't need to save the entire history or complete project in your own server	can take more time and space than usual
Merge conflicts are more	Merge conflicts are less

3. **Compare at least three configuration management tools. What are their similarities and differences?**

Configuration management tools

Puppet
Chef
Ansible
Saltstack

Similarities & Differences

Metrics	**Chef**	**Puppet**	**Ansible**	**Saltstack**
Availability	√	√	√	√
Ease of setup	Not very easy	Not very easy	Easy	Not very easy
Management	Not very easy	Not very easy	Easy	Not very easy
Scalability	Highly scalable	Highly scalable	Highly scalable	Highly scalable
Configuration	DSL(Ruby)	DSL(PuppetDSL)	YAML(Python)	YAML(Python)
Interoperability	High	High	High	High

4. **Explain the difference between continuous integration and continuous deployment.**

Continuous Integration is merging all code from all developers to one central branch of the repo many times a day trying to avoid conflicts in the code in the future. The concept here is to have multiple developers on a project to keep the main branch of the repo to the most current form of the source code, so each developer can check out or pull from the latest code to avoid conflicts.

Continuous Deployment, every change that is made is automatically deployed to production. This is a process that automatically deploys the results of Continuous Delivery into the final production environment, usually every time a developer changes code (assuming all automated tests pass). This practice requires excellent automated testing coverage and benefits from the ability to roll out changes slowly and to roll back changes quickly if something goes wrong.

5. **Using at least one example explains the difference between IAAS and PAAS cloud services.**

IaaS – Infrastructure as a Service
PaaS – Platform as a Service

IaaS customers can control their own data infrastructure without having to physically manage it on-site. Instead, they can access and store data on servers via a dashboard or API.

PaaS isn't delivering software over the internet, but it is providing an online platform that's accessible to different developers to create software delivered over the internet.

The most distinct difference between IaaS and PaaS is that IaaS offers administrators more direct control over operating systems, but PaaS offers users greater flexibility and ease of operation.

Let's say I wanted to start a website. I would need an IaaS product, like Amazon Web Services, to host it and its applications. If I wanted to create a custom feature, I could use a PaaS product like Google App Engine to design it and install it on my site.

6. **What is meant by "infrastructure as code". What are the advantages of this approach?**
   Infrastructure as code is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

   Advantages

   - Since it's just text, it's easy for you to edit, copy, and distribute it.
   - Infrastructure as code enables you to quickly set up your complete infrastructure by running a script. IaC can make the entire software development lifecycle more efficient.
   - You can guarantee the same configurations will be deployed over and over, without discrepancies like in manual infrastructure management. Provide consistency.
   - Accountability – since you can version IaC configuration files like any source code file, you have full traceability of the changes each configuration suffered.
   - Lower the costs of infrastructure management. That's because you won't have to spend money on hardware, hire people to operate it, and build or rent physical space to store it.

7. **What is meant by "staging" and "production" environments?**

   Staging: This is the release candidate, and this environment is normally a mirror of the production environment. The staging area contains the "next" version of the application and is used for final stress testing and client/manager approvals before going live.

   Production: This is the currently released version of the application, accessible to the client/end users. This version preferably does not change except for during scheduled releases.

8. **Briefly describe at least three metrics that are important to monitor in an application in production.**

   Host-Based Metrics

Towards the bottom of the hierarchy of primitive metrics are host-based indicators. These would be anything involved in evaluating the health or performance of an individual machine, disregarding for the moment its application stacks and services. These are mainly comprised of usage or performance of the operating system or hardware, like:

- CPU
- Memory
- Disk space
- Processes

These can give you a sense of factors that may impact a single computer's ability to remain stable or perform work.

Application Metrics

These are metrics concerned with units of processing or work that depend on the host-level resources, like services or applications. The specific types of metrics to look at depends on what the service is providing, what dependencies it has, and what other components it interacts with. Metrics at this level are indicators of the health, performance, or load of an application:

- Error and success rates
- Service failures and restarts
- Performance and latency of responses
- Resource usage

These indicators help determine whether an application is functioning correctly and with efficiency.

Network and Connectivity Metrics

For most types of infrastructure, network and connectivity indicators will be another dataset worth exploring. These are important gauges of outward-facing availability, but are also essential in ensuring that services are accessible to other machines for any systems that span more than one machine. Like the other metrics we've discussed so far, networks should be checked for their overall functional correctness and their ability to deliver necessary performance by looking at:

- Connectivity
- Error rates and packet loss
- Latency
- Bandwidth utilization

Monitoring your networking layer can help you improve the availability and responsiveness of both your internal and external services.

Server Pool Metrics

When dealing with horizontally scaled infrastructure, another layer of infrastructure you will need to add metrics for is pools of servers. While metrics about individual servers are useful, at scale a service is better represented as the ability of a collection of machines to perform work and respond adequately to requests. This type of metric is in many ways just a higher level extrapolation of application and server metrics, but the resources in this case are homogeneous servers instead of machine-level components. Some data you might want to track are:

- Pooled resource usage
- Scaling adjustment indicators
- Degraded instances

Collecting data that summarizes the health of collections of servers is important for understanding the actual capabilities of your system to handle load and respond to changes.

External Dependency Metrics

Other metrics you may wish to add to your system are those related to external dependencies. Often, services provide status pages or an API to discover service outages, but tracking these within your own systems—as well as your actual interactions with the service—can help you identify problems with your providers that may affect your operations. Some items that might be applicable to track at this level are:

- Service status and availability
- Success and error rates
- Run rate and operational costs
- Resource exhaustion

There are many other types of metrics that can be helpful to collect. Conceptualizing the most important information at varying levels of focus can help you identify indicators that are most useful for predicting or identifying problems. Keep in mind that the most valuable metrics on higher levels are likely to be resources provided by lower layers.