FullStack.Cafe - Kill Your Tech Interview

Q1: What is the need for DevOps? ☆

Topics: DevOps

Answer:

Nowadays instead of releasing big sets of features, companies are trying to see if small features can be transported to their customers through a series of release trains. This has many advantages like quick feedback from customers, better quality of software etc. which in turn leads to high customer satisfaction. To achieve this, companies are required to:

1. Increase deployment frequency
2. Lower failure rate of new releases
3. Shortened lead time between fixes
4. Faster mean time to recovery in the event of new release crashing

DevOps fulfills all these requirements and helps in achieving seamless software delivery.

Q2: What is the most important thing DevOps helps us achieve? ☆

Topics: DevOps

Answer:

The most important thing that DevOps helps us achieve is to get the changes into production as quickly as possible while minimising risks in software quality assurance and compliance. This is the primary objective of DevOps.

Q3: What is meant by *Continuous Integration*? ☆

Topics: DevOps

Answer:

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

Q4: Explain what is DevOps ? ☆

Topics: DevOps

Answer:

DevOps is a newly emerging term in IT field, which is nothing but a practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals. It focuses on delivering software product faster and lowering the failure rate of releases.

Q5: Are you more *Dev* or *Ops*? ☆

Topics: DevOps

Answer:

What the interview means is do you do more sysadmin work, or do you spend a lot of time working with developers on coding?

Q6: What is Kubernetes? Why organizations are using it? ☆

Topics: Kubernetes DevOps

Answer:

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

To understand what Kubernetes is good for, let's look at some examples:

- You would like to run a certain application in a container on multiple different locations. Sure, if it's 2-3 servers/locations, you can do it by yourself but it can be challenging to scale it up to additional multiple location.
- Performing updates and changes across hundreds of containers
- Handle cases where the current load requires to scale up (or down)

Q7: How is DevOps different from Agile/SDLC? ☆☆

Topics: DevOps

Answer:

- Agile software development methodology focuses on the development of software.
- DevOps on the other hand is responsible for development as well as deployment of the software in the safest and most reliable way possible.

Q8: Which are the top DevOps tools? Which tools have you worked on? ☆☆

Topics: DevOps

Answer:

The most popular DevOps tools are:

- **Git**: Version Control System tool
- **Jenkins**: Continuous Integration tool
- **Selenium**: Continuous Testing tool
- **Puppet, Chef, Ansible**: Configuration Management and Deployment tools
- **Nagios**: Continuous Monitoring tool
- **Docker**: Containerization tool

Q9: What are the *advantages* of DevOps? ☆☆

Topics: DevOps

Answer:

Technical benefits:

- Continuous software delivery
- Less complex problems to fix
- Faster resolution of problems

Business benefits:

- Faster delivery of features
- More stable operating environments
- More time available to add value (rather than fix/maintain)

Q10: What are the success factors for *Continuous Integration*? ☆☆

Topics: DevOps

Answer:

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

Q11: How have you handled *failed* deployments? ☆☆

Topics: DevOps

Answer:

Q12: Why is *Continuous Monitoring* necessary? ☆☆

Topics: DevOps

Answer:

Continuous Monitoring allows timely identification of problems or weaknesses and quick corrective action that helps reduce expenses of an organization. Continuous monitoring provides solution that addresses three operational disciplines known as:

- continuous audit

- continuous controls monitoring
- continuous transaction inspection

Q13: Mention what are the key aspects or principle behind DevOps? ☆☆

Topics: DevOps

Answer:

The key aspects or principle behind DevOps are:

- Infrastructure as code
- Continuous deployment
- Automation
- Monitoring
- Security

Q14: Can we consider DevOps as an Agile methodology? ☆☆

Topics: DevOps

Answer:

DevOps is a movement to reconcile and synchronize development and production start through a set of good practices . Its emergence is motivated by a deep changing demands of business, who want to speed up the changes to stick closer to the requirements of business and the customer.

Q15: What is DevOps engineer's duty with regards to Agile development? ☆☆

Topics: DevOps

Answer:

DevOps engineer work very closely with Agile development teams to ensure they have an environment necessary to support functions such as automated testing, continuous Integration and continuous Delivery. DevOps engineer must be in constant contact with the developers and make all required parts of environment work seamlessly.

Q16: What does *Containerization* mean? ☆☆

Topics: DevOps Docker

Answer:

Containerisation is a type of *virtualization* strategy that emerged as an alternative to traditional hypervisor-based virtualization.

In containerization, the operating system is shared by the different containers rather than cloned for each virtual machine. For example Docker provides a container virtualization platform that serves as a good alternative to hypervisor-based arrangements.

Q17: What is the function of *CI (Continuous Integration)* server? ☆☆

Topics: DevOps

Answer:

CI server function is to continuously integrate all changes being made and committed to repository by different developers and check for compile errors. It needs to build code several times a day, preferably after every commit so it can detect which commit made the breakage if the breakage happens.

Q18: What is the role of a *Configuration Management* tool in DevOps? ☆☆

Topics: DevOps

Answer:

Configuration Management tools' purpose is to automatize deployment and configuration of software on big number of servers. Most CM tools usually use agent architecture which means that every machine being manged needs to have agent installed.

One tool that uses agentless architecture is Ansible. It only requires SSH and Python. And if raw module is being used, not even Python is required because it can run raw bash commands. Other available and popular CM tools are Puppet, Chef, SaltStack.

Q19: What is post *Mortem Meetings*? ☆☆

Topics: DevOps

Answer:

Post mortem meeting is a meeting where we discuss what went wrong and what steps should be taken so that failure doesn't happen again. Post mortem meetings are not about finding the one to be blamed, they are for preventing outages from reoccurring and planing redesign of the infrastructure so that downtime can be minimised. It is about learning from mistakes.

Q20: What's the next thing you would automate in your current workflow? ☆☆

Topics: DevOps

Answer:

FullStack.Cafe - Kill Your Tech Interview

Q1: Explain *Blue-Green* deployment technique ☆☆☆

Topics: DevOps

Answer:

Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

This technique can eliminate downtime due to application deployment. In addition, blue-green deployment reduces risk: if something unexpected happens with your new version on Green, you can immediately roll back to the last version by switching back to Blue.

Q2: What's the difference between a *Blue/Green Deployment* and a *Rolling Deployment*? ☆☆☆

Topics: DevOps

Answer:

- In **Blue Green Deployment**, you have TWO complete environments. One is Blue environment which is running and the Green environment to which you want to upgrade. Once you swap the environment from blue to green, the traffic is directed to your new green environment. You can delete or save your old blue environment for backup until the green environment is stable.

- In **Rolling Deployment**, you have only ONE complete environment. The code is deployed in the subset of instances of the same environment and moves to another subset after completion.

Q3: How do all DevOps tools work together? ☆☆☆

Topics: DevOps

Answer:

Given below is a generic logical flow where everything gets automated for seamless delivery. However, this flow may vary from organization to organization as per the requirement.

1. Developers develop the code and this source code is managed by Version Control System tools like Git etc.
2. Developers send this code to the Git repository and any changes made in the code is committed to this Repository.
3. Jenkins pulls this code from the repository using the Git plugin and build it using tools like Ant or Maven.
4. Configuration management tools like Puppet deploys & provisions testing environment and then Jenkins releases this code on the test environment on which testing is done using tools like selenium.

5. Once the code is tested, Jenkins send it for deployment on the production server (even production server is provisioned & maintained by tools like Puppet).
6. After deployment It is continuously monitored by tools like Nagios.
7. Docker containers provides testing environment to test the build features.

Q4: What are the differences between *Continuous Integration*, *Continuous Delivery*, and *Continuous Deployment*? ☆☆☆

Topics: DevOps

Answer:

- Developers practicing **continuous integration** merge their changes back to the main branch as often as possible. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.
- **Continuous delivery** is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.
- **Continuous deployment** goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Q5: If something breaks in production, how do you know about it? ☆☆☆

Topics: DevOps

Answer:

Q6: What is *Chef*? ☆☆☆

Topics: DevOps

Answer:

Chef is a powerful automation platform that transforms infrastructure into code. Chef is a tool for which you write scripts that are used to automate processes.

- **Chef Server**: The Chef Server is the central store of your infrastructure’s configuration data. The Chef Server stores the data necessary to configure your nodes and provides search, a powerful tool that allows you to dynamically drive node configuration based on data.
- **Chef Node**: A Node is any host that is configured using Chef-client. Chef-client runs on your nodes, contacting the Chef Server for the information necessary to configure the node. Since a Node is a machine that runs the Chef-client software, nodes are sometimes referred to as “clients”.
- **Chef Workstation**: A Chef Workstation is the host you use to modify your cookbooks and other configuration data.

Q7: How would you assess how *deployable* a system is? ☆☆☆

Topics: DevOps

Answer:

Q8: How would you prepare for a migration from one platform to another? ☆☆☆

Topics: DevOps

Answer:

Q9: Tell me about the worst-run/best-run outage you've been a part of. What made it bad/well-run? ☆☆☆

Topics: DevOps

Answer:

Q10: How would you make key aspects of a software system *traceable*? ☆☆☆

Topics: DevOps

Answer:

Q11: What is the difference between *Resource Allocation* and *Resource Provisioning*? ☆☆☆

Topics: DevOps

Answer:

- Resource allocation is the process of reservation that demarcates a quantity of a resource for a tenant's use.
- Resource provision is the process of activation of a bundle of the allocated quantity to bear the tenant's workload.

Immediately after allocation, all the quantity of a resource is available. Provision removes a quantity of a resource from the available set. De-provision returns a quantity of a resource to the available set. At any time:

```
Allocated quantity = Available quantity + Provisioned quantity
```

Q12: Classify Cloud Platforms by *category* ☆☆☆

Topics: DevOps

Answer:

Cloud Computing software can be classified as:

- **Software as a Service or SaaS** - is peace of software that runs over network on remote server and has only user interface exposed to users, usually in web browser. For example salesforce.com.

- **Infrastructure as a Service or IaaS** - is a cloud environment that exposes VM to user to use as entire OS or container where you could install anything you would install on your server. Example for this would be OpenStack, AWS, Eucalyptus.
- **Platform as a Service or PaaS** - allows users to deploy their own application on the preinstalled platform, usually framework of application server and suite of developer tools. Examples for this would be Heroku.

Q13: What do you know about *Serverless* model? ☆☆☆

Topics: DevOps

Answer:

Serverless refers to a model where the existence of servers is hidden from developers. It means you no longer have to deal with capacity, deployments, scaling and fault tolerance and OS. It will essentially reducing maintenance efforts and allow developers to quickly focus on developing codes.

Examples are:

- Amazon AWS Lambda
- Azure Functions

Q14: Explain a use case for *Docker* ☆☆☆

Topics: DevOps Docker

Answer:

- Docker a low overhead way to run virtual machines on your local box or in the cloud. Although they're not strictly distinct machines, nor do they need to boot an OS, they give you many of those benefits.
- Docker can encapsulate legacy applications, allowing you to deploy them to servers that might not otherwise be easy to setup with older packages & software versions.
- Docker can be used to build test boxes, during your deploy process to facilitate continuous integration testing.
- Docker can be used to provision boxes in the cloud, and with swarm you can orchestrate clusters too.

Q15: What is the difference between Kubernetes and Docker? ☆☆☆

Topics: Docker Kubernetes DevOps

Problem:

And what are they used for?

Solution:

Docker and Kubernetes are complementary.

- **Docker** provides an open standard for packaging and distributing containerized applications, while
- **Kubernetes** provides for the orchestration and management of distributed, containerized applications created with Docker.

In other words, Kubernetes provides the infrastructure needed to deploy and run applications built with Docker.

Q16: Which problems does a *Container Orchestration* solve? ☆☆☆

Topics: Docker Kubernetes DevOps

Answer:

Containers run in an isolated process (usually in it's own namespace). This means that by default the container will not be aware of other containers. Additionally, it will not be aware of the systems files, network interfaces, and processes. While this can greatly help with portability of the software it does not solve several production issues such as microservices, container discovery, scalability, disaster recovery, or upgrades.

Adding a container orchestrator can greatly reduce the complexity in production as these tools are designed to resolve the issues outlined above. For example, Kubernetes is built to allow containers to be linked together, deploy containers across an entire network, scale and load balance the network based on container resource consumption, and allow upgrades of individual containers with no downtime.

If you are only running a single container or two containers together you are correct in that an orchestrator may be unnecessary and add unneeded complexity.

Q17: How would you deploy software to 5000 nodes? ☆☆☆☆

Topics: DevOps

Answer:

Q18: How is *Container* different from a *Virtual Machine*? ☆☆☆☆

Topics: DevOps Docker

Answer:

- Unlike a virtual machine, a container does not need to boot the operating system kernel, so containers can be created in less than a second. This feature makes container-based virtualization unique and desirable than other virtualization approaches.
- Since container-based virtualization adds little or no overhead to the host machine, container-based virtualization has near-native performance
- For container-based virtualization, no additional software is required, unlike other virtualizations.
- All containers on a host machine share the scheduler of the host machine saving need of extra resources.
- Container states (Docker or LXC images) are small in size compared to virtual machine images, so container images are easy to distribute.
- Resource management in containers is achieved through cgroups. Cgroups does not allow containers to consume more resources than allocated to them.

Q19: What is *Vagrant* and what is it used for? ☆☆☆☆

Topics: DevOps

Answer:

Vagrant is a tool that can create and manage virtualized (or containerized) environments for testing and developing software. At first, Vagrant used virtualbox as the hypervisor for virtual environments, but now it supports also KVM.

Q20: What is *Continuous Monitoring*? ☆☆☆☆

Topics: DevOps

Answer:

Continuous monitoring gets into the depth of monitoring coverage, from in-browser front-end performance metrics, through application performance, and down to host virtualized infrastructure metrics.

FullStack.Cafe - Kill Your Tech Interview

Q1: Why *Continuous Integration* is important for Agile? ☆☆☆☆

Topics: Agile & Scrum DevOps

Answer:

Continuous Integration is important for Agile for following reasons:

- It helps to **maintain release schedule** on time by detecting bugs or integration errors
- Due to frequent agile code delivery usually every sprint of 2-3 weeks, stable quality of build is a must and continuous integration ensures that
- In helps to maintain the **quality and bug free state of code-base**
- Continuous integration helps to** check the impact of work on branches** to the main trunk if development work is going on branches using automatic building and merging function

Q2: How would you introduce Continuous Delivery in a successful, huge company for which the change from Waterfall to Continuous Delivery would be not trivial, because of the size and complexity of the business? ☆☆☆☆☆

Topics: DevOps

Answer:

Q3: What is *Canary Releasing*? ☆☆☆☆☆

Topics: DevOps

Answer:

Canary Releasing is a technique to reduce the risk of introducing a new software version in production. This is done by slowly rolling out the change to a small subset of users before giving it out to the entire infrastructure, i.e. making it available to everybody.

Q4: Can you explain a relationship between *container runtime* and *container orchestration*? ☆☆☆☆☆

Topics: Docker Kubernetes DevOps

Answer:

- A **container runtime** is the part of your container environment in charge of the creation and basic features of your containers. The obvious one is Docker, but you can also find Containerd, CRI-O and other as runtime for your containers.

- An **orchestrator**, by contrast, will not exactly create your container (ie, an orchestrator is not the technology used to create them). An orchestrator will use your container runtime to manage them. In a way,

it's a layer around your container runtime wich allow finer usages, such as scaling, multi host deployments, load balancing...

The most famous orchestrators would be Kurbenetes and Docker Swarm.