

TeamViewer - Client Operations Engineer - Task



Solution provided by:

Christiana Chinwe Onyebueke

Motivated cloud engineer with experience designing, deploying, and maintaining public and hybrid cloud infrastructures focusing on reliability, best practices, cost optimization, security, performance, and operational excellence. Versatile, meticulous, customer-centric, team player, and strategic thinker with excellent time and prioritization management skills. Quick learner with a “can-do” attitude, and passionate about cloud IT innovation, and process improvement. Open to applicable opportunities to leverage my skill set in a team that believes in achieving excellence and in winning as a team.



Question 1

Explain the difference between Docker and virtual machines (VMs) and when you would choose one over the other for deploying applications.



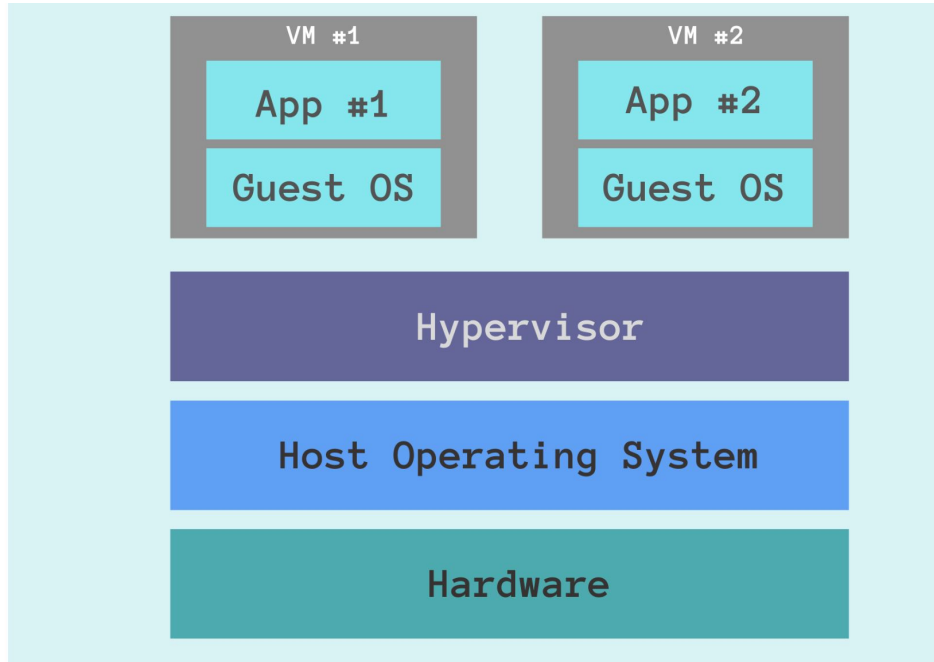
VM

Virtual Machine Mechanism



- A Virtual Machine or VM is the emulation of a physical computer inside a host machine.
- A hypervisor, a software layer running on the host operating system, manages virtual machine (VM) instances.
- Each VM operates with its guest operating system, providing an isolated environment for applications to run.
- It's possible to have multiple VMs, each hosting a different application and operating system.

Here is example of Diagram explaining how VM operates in the cloud



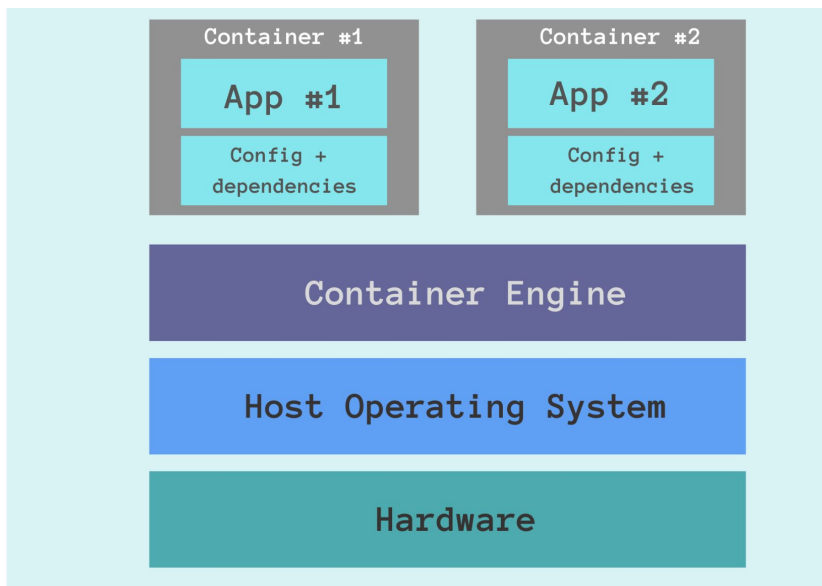
Docker Container Mechanism




- A Docker container is a lightweight, portable software package that includes an application along with its dependencies and configuration settings.
- Docker containers are like shipping containers: they standardize packaging. Instead of goods, they carry software and its dependencies, ensuring consistent performance across environments.
- Unlike virtual machines, Docker containers don't launch a separate guest operating system. Instead, they run directly on the host OS, made possible by a container engine.

Explain the difference between Docker and virtual machines (VMs) and when you would choose one over the other for deploying applications

Here is example of Diagram explaining how Docker operates in the cloud



Difference between Docker and virtual machines (VMs)



	Docker	Virtual Machines (VMs)
Architecture	Containers share the host OS kernel.	Each VM includes a full OS and a virtualized hardware stack.
Resource Utilization	Lightweight, minimal overhead.	Resource-heavy due to full OS and hardware emulation.
Startup Time	Fast (seconds).	Slow (minutes).
Isolation	Process-level isolation with shared OS.	Full OS-level isolation.
Security	Less secure; potential risks from shared kernel.	More secure; complete isolation.

Explain the difference between Docker and virtual machines (VMs)



	Docker	Virtual Machines (VMs)
Portability	Highly portable across environments supporting containers.	Less portable; tied to hypervisor and host hardware.
Performance	High performance, minimal resource usage.	Lower performance due to virtualization overhead.
Use Cases	Microservices, lightweight applications, CI/CD pipelines.	Legacy apps, OS-specific workloads, strict isolation needs.
Scalability	Scales quickly and efficiently.	Slower scaling; resource-heavy.
Dependencies	Requires a container runtime (e.g., Docker Engine).	Requires a hypervisor (e.g., VMware, Hyper-V, KVM).

When to Choose Docker over VMs



I will choose Docker over a VMs for application deployment if my application:

- **Requires fast and efficient scaling** – Containers can be spun up or shut down quickly, making it easy to scale applications dynamically based on demand. Unlike VMs, which require booting an entire OS, containers start almost instantly, reducing deployment time.
- **Requires smooth integration with CI/CD pipelines** – Docker ensures a uniform environment across development, testing, and production, eliminating inconsistencies often encountered between different stages. This facilitates automated builds, testing, and deployments, enhancing both development efficiency and reliability.
- **Follows a microservices architecture** – If my application consists of multiple independent services that need to be deployed, updated, and scaled separately, Docker allows each microservice to run in its own container while communicating efficiently with others.
- **Must run consistently across multiple environments** – Containers encapsulate everything the application needs (dependencies, libraries, configurations), ensuring that it runs identically on a developer's laptop, staging, and production environments without compatibility issues.

When to Choose Docker over VMs



I will choose Docker over a VMs for application deployment if my application:

- **Needs rapid rollbacks and updates** – Docker allows for version control, making it easy to roll back to a previous version if a deployment introduces issues. Updates can be deployed seamlessly by replacing old containers with new ones.
- **Is designed for cloud-native deployment** – Many modern applications are built to run on container orchestration platforms like Kubernetes, AWS ECS, or Google Cloud Run. Docker provides a standardized way to deploy and manage applications efficiently in cloud environments.
- **Must operate in a resource-constrained environment** – Unlike VMs, which require dedicated OS instances, Docker containers share the host OS, consuming fewer system resources and allowing for better efficiency when running multiple applications on the same hardware.

When to Choose VMs over Docker



I will choose **VMs** over Docker for application deployment if my application:

- **Requires strong isolation and security** – VMs provide full OS-level isolation, making them ideal for multi-tenant environments where security is a priority. Unlike Docker, where containers share the host OS kernel, VMs run entirely separate OS instances, reducing the risk of security vulnerabilities from shared resources.
- **Needs to run different operating systems** – If my application requires multiple OS environments (e.g., running both Windows and Linux applications), VMs allow each instance to have its own dedicated OS, whereas Docker containers must use the same OS kernel as the host.
- **Hosts legacy or monolithic applications** – Many legacy applications are not designed to run in containers and require specific system configurations, dependencies, or traditional deployment models that work better on VMs. Monolithic applications that don't follow microservices principles may also perform better in a VM environment.
- **Requires full hardware virtualization** – If my application needs direct access to physical hardware resources like GPUs, network interfaces, or USB devices, VMs provide better support since they virtualize the entire machine, whereas Docker operates in user space with limited hardware access.

When to Choose VMs over Docker



I will choose VMs over Docker for application deployment if my application:

- **Has complex networking or storage requirements** – Applications that require advanced networking setups, persistent storage, or specialized configurations (such as specific disk partitions, RAID setups, or SAN storage) are often better suited for VMs, which provide more control over infrastructure-level configurations.
- **Demands long-running stability over rapid deployment** – VMs are ideal for applications that need to run continuously for extended periods without frequent updates. Unlike containers, which are designed for rapid changes and scaling, VMs provide a more stable and predictable environment for mission-critical workloads.
- **Faces software licensing or compliance constraints** – Some enterprise applications have licensing models that restrict running in a containerized environment. Regulatory requirements in industries like finance or healthcare may also mandate full OS separation, making VMs the better choice.



Question 2

Explain the concept of Infrastructure as Code (IaC) and its benefits.

INFRASTRUCTURE

AS CODE

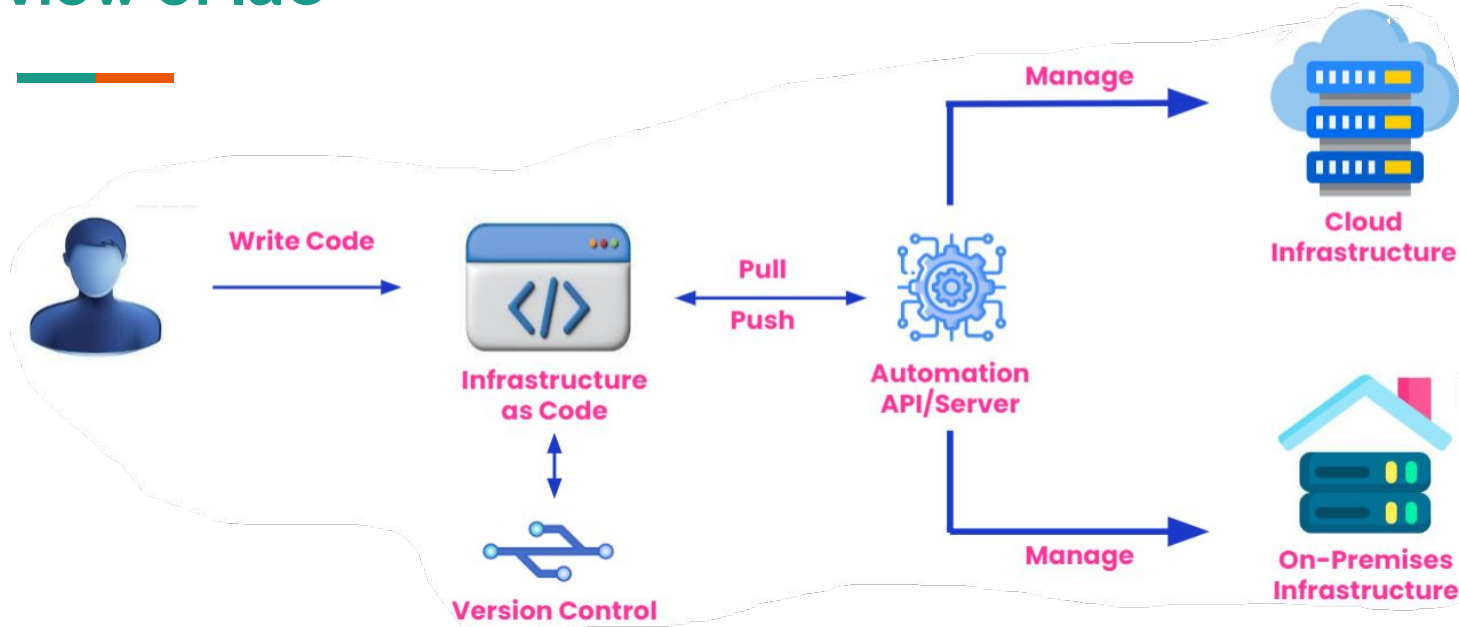


What is Infrastructure as Code (IaC)?



- Infrastructure as Code (IaC) is the practice of managing and provisioning computing infrastructure through machine-readable files, allowing automation and consistency across environments.
- IaC enables the application of software development practices, such as version control and testing, to infrastructure management, resulting in more reliable, scalable, and efficient operations.

Overview of IaC



This diagram shows the process of **Infrastructure as Code (IaC)**, where users first **write code** to define their infrastructure. The code is then placed under **version control** to track changes. Afterwards, the code is either **pulled or pushed** through an **automation API/server** to provision and manage resources. Finally, it automates the management of both **cloud** and **on-premises** infrastructure efficiently.

Examples of IaC Software.



ANSIBLE



Chef Server

Key Benefits of IaC



- Defines infrastructure in versioned, human-readable configuration files
- Enables automation and consistency across environments.
- Reduces manual errors by automating infrastructure management.
- Facilitates infrastructure changes and scaling through code updates.

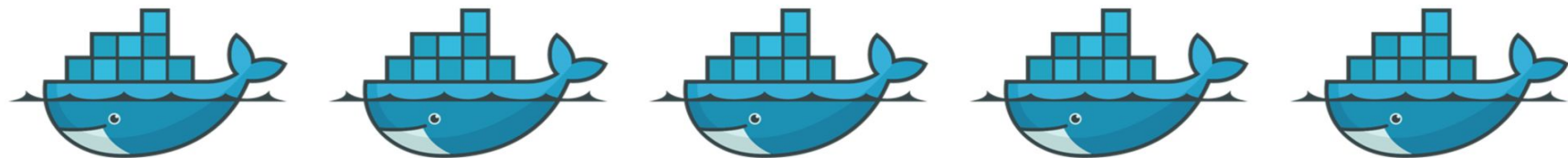


Question 3

What is Kubernetes, and how does it facilitate container orchestration in a DevOps environment?



kubernetes



docker docker docker docker docker

What is Kubernetes?



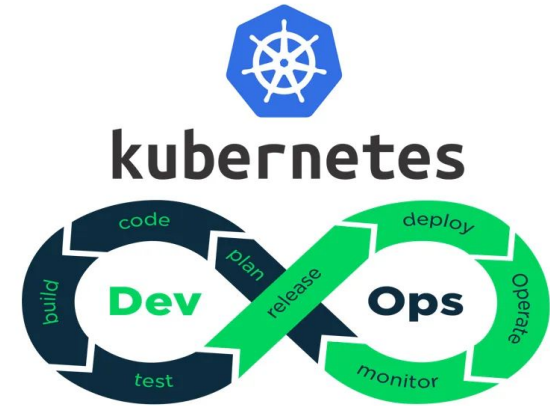
Kubernetes is an open-source platform designed to manage containerized workloads and services. It is portable, extensible, and helps automate deployment and configuration of containers. Kubernetes provides a framework to run distributed systems with high resilience, scalability, and failover management.

Why you need Kubernetes?



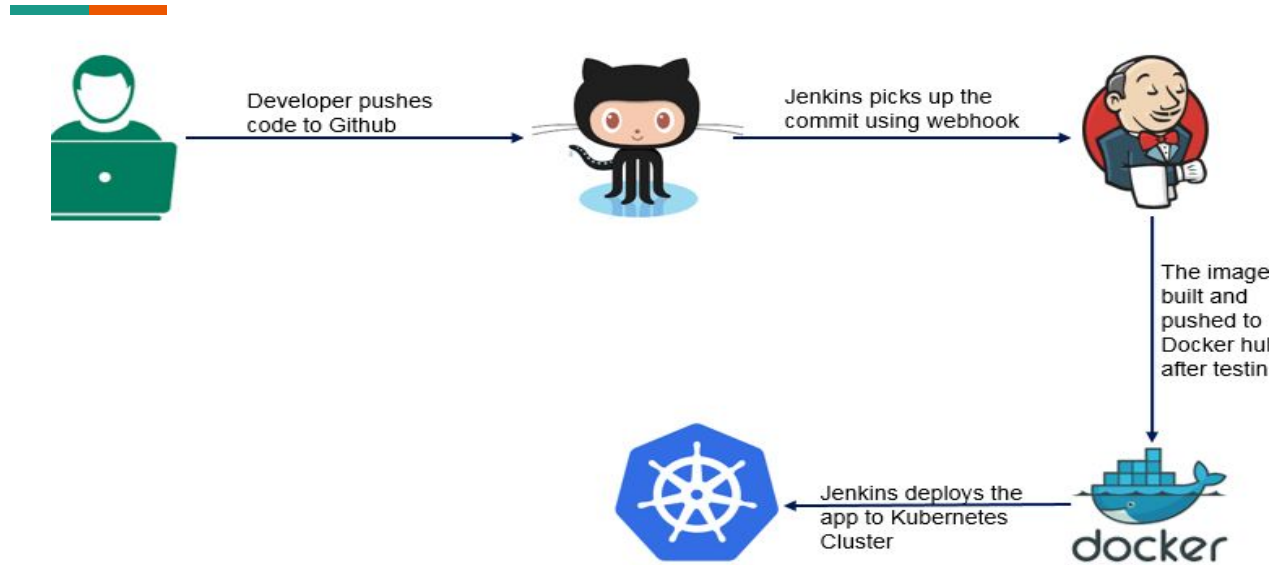
- Kubernetes is essential for efficiently managing containers in production environments, especially when ensuring high availability and reliability.
- It automates critical tasks like scaling applications and handling failover, so if a container fails, Kubernetes can quickly replace it with another, ensuring continuous service.
- Kubernetes also provides features like load balancing, service discovery, and the orchestration of storage systems.
- It enables automated rollouts and rollbacks of applications, adjusting containers to meet desired specifications.
- It offers self-healing capabilities, automatically restarting or replacing containers that fail health checks.
- Simplifies secret and configuration management, supports horizontal scaling based on demand, and allows for dual-stack IPv4/IPv6 networking.
- Kubernetes is highly extensible, allowing users to add new features without altering its core source code.

how does kubernetes facilitate container orchestration in a DevOps environment?



- In DevOps environment, kubernetes streamlines container orchestration by automating crucial processes like deployment, scaling, fault tolerance, and monitoring.
- It facilitates quicker and more dependable software delivery, encourages teamwork, supports a variety of deployment techniques, and maximizes resource use while preserving high availability. As a result, the DevOps pipeline becomes more flexible and effective, enabling businesses to innovate continuously while preserving a reliable production environment.
- It supports essential DevOps approaches like CI/CD, Infrastructure as Code, and automated recovery. DevOps teams may now concentrate on producing dependable, high-quality applications quickly.

Automated Code Deployment with Jenkins, Docker, and Kubernetes



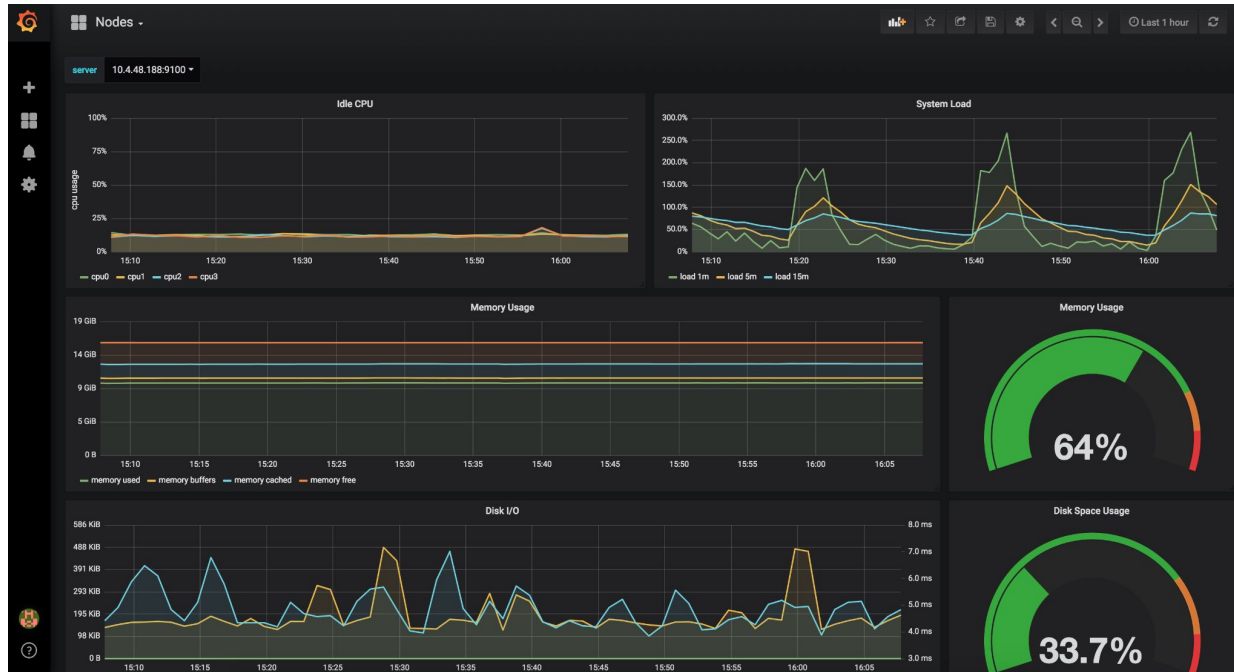
The diagram shows a developer pushing code to GitHub, initiating the process. Jenkins picks up the commit through a webhook, which triggers an automated pipeline. Jenkins then builds the application image based on the latest commit. The built image is pushed to Docker Hub, making it available for deployment. Finally, Jenkins deploys the updated application to a Kubernetes cluster, ensuring that the new version is live and operational in the containerized environment.



Question 4

What is Prometheus, and how does it contribute to monitoring and alerting in a DevOps environment?

Prometheus



What is Prometheus



- Is an open-source monitoring framework with high scalability capabilities and it offers unconventional monitoring features for the kubernetes orchestration platform.
- It collects and stores time-series data (metrics) from configured targets such as services, applications, and infrastructure components, providing robust querying and long-term trend analysis.
- Prometheus's labeled metrics provide for a more adaptable and scalable method of gathering and querying data across several dimensions, including host, service, and status.
- Prometheus provides PromQL, an expressive and adaptable query language, for retrieving and analyzing metrics, which produces insights into the performance of the system.
- Prometheus mostly collects metrics using a pull-based paradigm, which involves periodically scraping endpoints to collect information.
- When specified conditions are fulfilled, such as when CPU consumption above a predetermined limit, Prometheus can send notifications based on user-defined thresholds by integrating with an alert manager.
- Prometheus facilitates dynamic environments by automatically locating and keeping an eye on services in cloud environments, Kubernetes, and other infrastructure.

How does it contribute to monitoring and alerting in a DevOps environment?



- Prometheus keeps an eye on infrastructure, programs, and systems all the time, gathering data on things like request latency, CPU utilization, memory usage, and error rates. Teams are better able to comprehend the state of their services due to this real-time monitoring.
- Prometheus makes automated alerting possible with Alertmanager. The DevOps team is notified via alerts when performance problems or possible outages are identified (such as high error rates or low resource availability), or they can link with other communication platforms (e.g., Slack, email, etc.). This facilitates the proactive solving of problems.
- Prometheus plays a key role in debugging and troubleshooting because of its capacity to retain previous metrics and run comprehensive queries using PromQL. DevOps teams are able to spot performance snags, analyze patterns, and spot irregularities in their systems.

How does it contribute to monitoring and alerting in a DevOps environment?



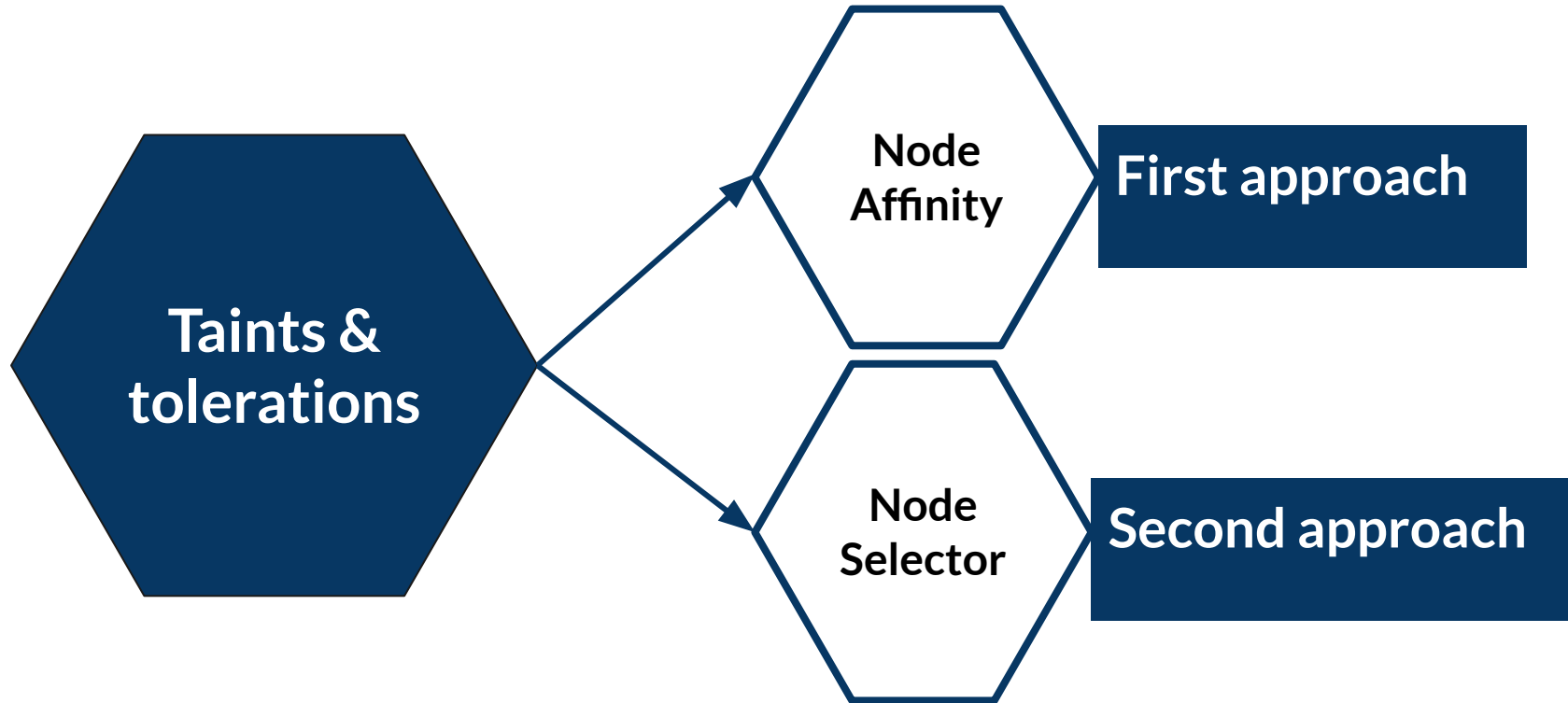
- Monitoring across dynamic, distributed systems is made possible by Prometheus's high scalability and seamless integration with cloud-native architectures and Kubernetes. It assists in making sure that every element functions properly in DevOps environments, particularly in intricate, microservice-driven systems.
- Prometheus assists teams in identifying opportunities for optimization and enhancement by providing insight into application performance. This helps create a constant feedback loop, which is crucial to the DevOps culture and ensures that performance can be monitored and enhanced over time.



Question 5

How would you manage images to be deployed only in a certain nodepool?

How would you manage images to be deployed only in a certain nodepool?



Taints & Tolerations and Node Affinity rules



- In order to fully dedicate nodes for particular pods following the best practices, I will be incorporating the Taints & Tolerations and Node Affinity rules.
- **Taint**, for instance, can be used to designate a node as requiring "maintenance" and stop pod scheduling on the node during that time.
- **Tolerations** allow pods to schedule on tainted nodes.
- Based on node labels, **node affinity** will be utilized to determine which nodes a pod should or shouldn't be scheduled on.
- *Compared to node selector, node affinity offers more precise control over pod scheduling and enables you to define intricate criteria for scheduling pods based on numerous node labels.*

Taints & Tolerations and Node Affinity rules

1. Apply a taint to the nodes in your specific node pool. You can achieve this by using the `kubectl taint` command. So, I want to taint a node with key "disktype" and value "ssd".
2. You can include tolerance in the pod specification to allow for a taint. By way of example, the pod specification that follows has a 3600-second tolerance time for the taint "disktype=ssd":

```
kubectl taint nodes <node-name>  
disktype=ssd:NoSchedule
```

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: my-pod  
spec:  
  tolerations:  
    - key: "disktype"  
      operator: "Equal"  
      value: "ssd"  
      effect: "NoSchedule"  
      tolerationSeconds: 3600  
  containers:  
    - name: my-container  
      image: nginx
```

Node Affinity (second approach)



Based on node labels, node affinity enables users to designate which nodes a pod should or shouldn't be scheduled on. Node affinity, for instance, can be used to indicate that a pod should be scheduled on a node that has a particular label, like `disktype=ssd`.

Use the `affinity` field and set the `nodeAffinity` field to the required node affinity rules in order to specify node affinity in a pod definition. For instance, the pod should be scheduled on a node with the label `disktype=ssd`, according to the following pod specification:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnor
      edDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
        containers:
          - name: my-container
            image: nginx
```

Node selector (Third approach)

Similar to node affinity, node selector lets users designate which nodes a pod should be scheduled on using node labels. But in contrast to node affinity, which offers more precise control over pod scheduling, node selector is a more basic and straightforward mechanism

Kubernetes' node selective functionality lets users choose which nodes, based on node labels, a pod should be scheduled on. Adding labels to nodes and then defining the preferred node labels in the pod specification is how the node selector operates.

The `kubectl label` command is used to add labels to a node. For instance, run the following command to give a node the label `disktype=ssd`:

```
kubectl label nodes <node-name>  
disktype=ssd
```

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: my-pod  
spec:  
  nodeSelector:  
    disktype: ssd  
  containers:  
  - name: my-container  
    image: nginx
```

Taints & Tolerations and Node Affinity rules



- In order to fully dedicate nodes for particular pods following the best practices, I will be incorporating the Taints & Tolerations and Node Affinity rules.
- Taint, for instance, can be used to designate a node as requiring "maintenance" and stop pod scheduling on the node during that time.
- **Tolerations** allow pods to schedule on tainted nodes.
- Based on node labels, node affinity will be utilized to determine which nodes a pod should or shouldn't be scheduled on.
- Compared to node selector, node affinity offers more precise control over pod scheduling and enables you to define intricate criteria for scheduling pods based on numerous node labels.

Node selector should be used when you want to specify which nodes a pod should be scheduled on based on node labels, but do not need the fine-grained control provided by node affinity.



Question 6

How would you manage images to be deployed only in a certain nodepool?



Question 6

The 1.25 release series includes Kubernetes version 1.25.15-gke.1115000. The Kubernetes release history indicates that the 1.25 series ended on October 28, 2023.

KUBERNETES.IO

This indicates that maintenance support, including security patches and bug fixes, is no longer available for version 1.25.15-gke.1115000 as of that date.

It is noteworthy that despite the upstream Kubernetes end-of-life dates, Google Kubernetes Engine (GKE) might still support specific versions. However, version 1.25.15-gke.1115000 is regarded as out of date as of February 8, 2025. Upgrading to a more recent GKE version is advised to guarantee that your clusters stay safe and maintained.



Question 7

We have an application that required images pulled from 3 different repo online. How would you go about packaging that so it is able to be installed on a closed gap cluster.