Migrating from VMs to Containers with Kubernetes: A Step-by-Step Guide

Naviteq

Migrating from VMs to Containers with Kubernetes: A Step-by-Step Guide

The landscape of application deployment has undergone a significant transformation in recent years. Traditional virtual machines (VMs), while reliable, are increasingly giving way to container-based architectures that offer unprecedented flexibility and efficiency. This guide walks you through the process of migrating from virtual machines to containers using Kubernetes.

Even though virtual machines have been the backbone of enterprise computing for decades, providing isolation, resource allocation, and the ability to run multiple environments on a single physical server, containers represent a more lightweight and agile approach to application deployment.

Key advantages of containerization are:

- **Lightweight footprint:** Containers share the host OS kernel, consuming far less resources compared to full VMs.
- **Consistent environments:** Containers ensure identical deployment across development, staging, and production environments.
- **Rapid scaling:** Quickly scale up or down application instances based on demand.
- **Improved resource utilization:** More instances of an application can be deployed on the same server.
- **Simplified dependency management:** Applications are packaged with their required dependencies, thus making a single small cohesive deployable unit.

Naviteq

Understanding Virtual Machines & Containers

What are Virtual Machines?

Virtual Machines are software-defined environments that emulate physical computers. They operate by creating an isolated, virtualized layer on top of a physical host machine's hardware. This virtualization layer is managed by a software called hypervisor. Hypervisor allocates and manages the host's physical resources like CPU, memory, storage, and network interfaces among the virtual machines. Each VM has its own virtual hardware, including a virtual CPU, virtual memory, virtual disks, and virtual network adapters.

What are containers?

Containers are lightweight, portable units designed to package an application along with all its dependencies, configurations, and libraries. This enables consistent deployment across different computing environments. Unlike traditional virtual machines, containers share the host operating system's kernel, making them more efficient and faster to start up. This approach to application packaging ensures that the application runs reliably and in the exact same manner whether it’s running in a testing environment, in production or locally on a developer’s system. Containers achieve this consistency by bundling everything needed to run the application. The code, runtime, system tools, system libraries, and settings are all bundled into a single self-contained package.

Naviteq

Comparing VMs and containers?

Characteristic	VMs	Containers
Portability	Limited	Excellent
Scalability	Slow	Rapid
Isolation	High	Limited
Resource utilization	Less efficient	Very efficient
Boot time	Slower	Quicker

If you want a more detailed comparison between containers and virtual machines, checkout our blog post on the topic.

What is Kubernetes?

Kubernetes is an extensible open source system which is used to deploy, scale and manage containerized applications. As containers are a widely adopted technology in the modern DevOps ecosystem, a tool is needed to manage these container deployments effectively. Kubernetes solves this issue and is the leading tool in container orchestration space. It automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes, scaling up and down and monitoring your applications, thus making application management easy and streamlined.

What are the features of Kubernetes?

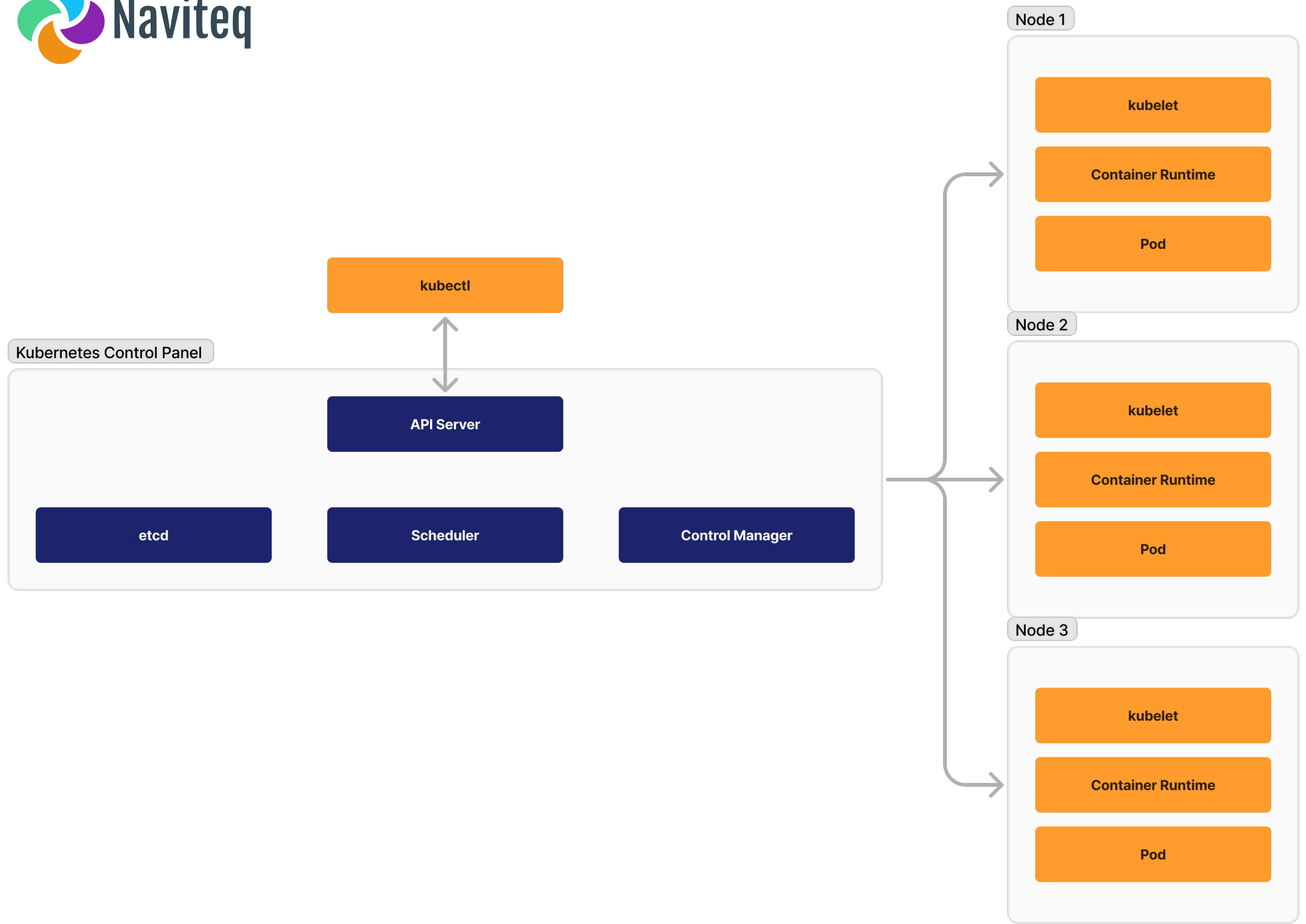
- **Automated rollouts and rollbacks:** Kubernetes supports zero-downtime updates by gradually introducing new versions of applications and provides the ability to automatically roll back to previous versions if any errors are detected, minimizing risks associated with software updates.
- **Self-healing:** Kubernetes automatically replaces and reschedules containers that fail, kills unresponsive containers, and maintains continuous application availability by constantly monitoring and correcting the state of deployed services.
- **Service discovery and load balancing:** It automatically assigns IP addresses to containers and intelligently distributes network traffic, creating a stable and responsive deployment environment that simplifies inter-service communication.
- **Scaling:** Kubernetes scales your application up and down with a simple command, with a UI, or automatically based on CPU usage. Kubern
- **Secret and configuration management:** It securely stores and manages sensitive information, allowing teams to update configurations without rebuilding container images and ensuring that critical data remains encrypted and protected.

Naviteq

Structure of a container deployment using Kubernetes

A container deployment using Kubernetes has these general components:

- **kubectl:** Command-line interface tool used by DevOps engineers for interacting with and managing Kubernetes clusters.
- **API Server:** API Server is the central control hub that exposes the Kubernetes API and handles all administrative tasks and communication between components.
- **etcd:** It is a key-value store used as Kubernetes' primary database for storing cluster configuration and state.
- **Scheduler:** The scheduler assigns newly created pods to nodes based on resource requirements, affinity/anti-affinity rules, and available computing resources.
- **Control manager:** It runs the controller processes that regulate the state of the cluster, ensuring the actual state matches the desired state.
- **Node:** A server instance that runs containerized applications in Kubernetes.
- **Container Runtime:** The tool responsible for running containers. Some common container runtimes are Docker, containerd and CRI-O.
- **kubelet:** Kubernetes agent running on each node that ensures containers are running in a pod.
- **Pod:** Pod is the smallest deployable unit in Kubernetes. It can contain one or more containers that share resources.


kubectl
Kubernetes Control Panel
API Server
etcd
Scheduler
Control Manager
Node 1
kubelet
Container Runtime
Pod
Node 2
kubelet
Container Runtime
Pod
Node 3
kubelet
Container Runtime
Pod

Pre-migration considerations

1. Assess applications

1. Identify applications suitable for containerization (stateless microservices are ideal).
2. Evaluate application dependencies (complex OS dependencies might require refactoring).
3. Assess performance requirements and potential impact of containerization.

2. Define goals

Set clear objectives for migration (e.g., faster deployments, improved scalability, reduced costs).

3. Select tools

- **Docker:** Container creation and management.
- **Kubernetes:** Container orchestration platform.
- **Helm:** Package manager for Kubernetes deployments (charts).
- **CI/CD Tools:** Integrate container builds and deployments into your CI/CD pipeline (e.g., Jenkins, GitLab CI, GitHub Actions).

Naviteq

Preparing the environment

1. Infrastructure and dependencies

1. Ensure your infrastructure can support containerized workloads i.e it has sufficient CPU, memory and network bandwidth.
2. Choose a container-optimized operating system (OS) for your Kubernetes nodes.
3. Configure your network to support container networking e.g., CNI plugins.
4. Select a storage solution compatible with Kubernetes persistent volumes for stateful applications.

2. Container runtime

Choose a container runtime like Docker, containerd, CRI-O, or Podman to execute container images on your nodes.

3. Kubernetes setup

1. Decide on a managed Kubernetes services such as AKS, EKS, GKE or a self-hosted deployment.
2. Configure cluster networking using flannel, weave net etc.
3. Set up persistent storage provisioning. Some common options are hostPath, NFS, iSCSI.
4. Implement role-based access control (RBAC) for secure access to the cluster.
5. Configure monitoring and logging for your Kubernetes cluster using tools like Prometheus, Grafana etc.

Naviteq

Step-by-step migration process

Step 1: Plan the migration

1. Prioritize applications for containerization.
2. Create a detailed inventory of applications, including:
    - Runtime environments
    - Base operating systems
    - Configuration files
    - Environment variables
    - Storage requirements
    - Network dependencies
3. Categorize applications by complexity and migration difficulty:
    - Stateless web applications are generally easier to containerize than those with shared file systems.
4. Develop a phased migration roadmap with milestones:
    - Setting up Kubernetes.
    - Establishing container registry and image management.
    - Creating base container images and Dockerfile templates.
    - Implementing container security scanning, monitoring, and logging.
    - Training teams on container technologies and best practices.

Step 2. Create container images

1. Use Dockerfiles to define the environment and dependencies for each application.
2. Leverage multi-stage builds for efficient image creation:
    - Separate the build environment from the runtime environment to minimize image size.practices.

3. Minimize image size:
   - Remove unnecessary files, caches, and temporary data.
   - Use .dockerignore to exclude irrelevant files from the build context.
   - Chain RUN commands with && to reduce layer count.
   - Use minimal base images (e.g., slim or alpine) when appropriate.
   - Only install required packages and dependencies.

4. Implement best practices for image security:
   - Run containers as non-root users.
   - Set filesystem and volume permissions appropriately.
   - Regularly update base images to patch vulnerabilities.
   - Use COPY instead of ADD to prevent remote file injection.
   - Specify exact versions of base images and dependencies.
   - Implement resource limits and constraints.
   - Scan images for vulnerabilities using tools like Trivy or Snyk.
   - Use proper secrets management solutions.

5. Leverage official base images when possible for consistency and security.

Step 3: Test containerized applications

1. Thoroughly test containerized applications in a staging environment to identify and address issues proactively.

2. Perform unit testing to verify isolated functionality of containerized components.

3. Conduct integration testing to ensure containerized applications work together as expected.

4. Implement performance benchmarking to compare containerized vs. non-containerized versions.

5. Perform security scanning of container images to identify vulnerabilities.

6. Conduct compatibility testing across different environments (OS, cloud providers, storage classes).

Step 4: Implement orchestration with Helm

Use Helm charts to package and deploy applications to Kubernetes. To implement orchestration with Helm, you'll first need to create a chart structure which includes a Chart.yaml file that defines metadata, a values.yaml file for configurable parameters, and templates directory containing Kubernetes manifest templates.

The process of creating a basic chart structure is:

- Initialize a new chart with **helm create chartname.**
- Define chart metadata in Chart.yaml including dependencies.
- Create default values in **values.yaml.**
- Include **README.md** with usage instructions and configuration details.

Step 5: Deploy to production

Deployment strategies

- Define your deployment strategy and implement it using CI\CD tools (GitHub actions, ArgoCD etc.).
- Test your deployments 24/7, especially during and after the new version rollouts.

Robust monitoring

- **Kubernetes Metrics Server:** Collects metrics about cluster resources, nodes, pods, and containers. This should only be used for autoscaling purposes.
- **Prometheus:** Monitor custom metrics and alerts using Prometheus.
- **Grafana:** Visualize metrics and create custom dashboards using Grafana.
- **Jaeger:** Implement distributed tracing to track requests across microservices using Jaeger.

Autoscaling

- **Horizontal Pod Autoscaler (HPA):** Automatically scales the number of pods based on CPU utilization or custom metrics.
- **Cluster autoscaler:** Automatically scales the number of nodes in the cluster.
- **Vertical Pod Autoscaler (VPA):** Automatically adjusts resource requests and limits for pods.

Centralized logging

- **Log rotation:** Configure log rotation policies to manage disk space.
- **Log aggregation:** Centralize logs from all nodes and pods.
- **Log analysis:** Use tools like Logstash to parse and analyze logs.

Rollback mechanisms

- **Kubernetes rollback:** roll back deployments to previous versions in case of issues.
- **Configuration management:** Use tools like Helm or Kustomize to manage configuration and roll back changes.
- **Automated rollback triggers:** Set up automated rollbacks based on monitoring metrics and alerts.

Key challenges and solutions

Handling state and storage

- **Persistent Volumes (PVs) and Persistent Volume Claims (PVCs):** Use PVs to provision storage resources and PVCs to request storage from PVs and ensure data persistence across pod restarts and node failures.

- **StatefulSets:** Manage stateful applications that require stable storage and network identities. It's ideal for databases, message queues, and other stateful services.

- **External storage solutions:** Integrate with cloud-native storage providers like AWS EBS, GCP Persistent Disk, and Azure Disk Storage using CSI drivers to leverage features like dynamic provisioning snapshots and replication.

Security considerations

- **Image scanning:** Use tools like Trivy or Clair to scan container images for vulnerabilities and ensure images are up-to-date and free of known vulnerabilities.

- **Network policies:** Define network policies to control communication between pods and external networks. Limit network exposure and prevent unauthorized access.

- **Secrets management:** Use Kubernetes Secrets to securely store sensitive information. You should also consider external secrets management tools like HashiCorp Vault and/or External Secrets for advanced features.

- **Role-Based Access Control (RBAC):** Implement RBAC to control access to Kubernetes resources. Grant permissions based on user roles and responsibilities.

Naviteq

Managing complexity with Kubernetes tools

- **Helm:** Package and deploy complex applications to Kubernetes using Helm. It manages application dependencies, configurations, and upgrades.

- **ArgoCD:** Implement GitOps workflows for automated deployments and rollbacks using ArgoCD. It can be used to sync Kubernetes manifests with Git repositories.

- **Prometheus and Grafana:** Monitor Kubernetes clusters and applications proactively using tools like Prometheus and visualize metrics and create custom dashboards using Grafana.

Additional considerations

- **Kubernetes operators:**
  - Automate the management of complex applications and infrastructure.
  - Provide lifecycle management, configuration, and scaling.

- **Service mesh:**
  - Simplify service-to-service communication and improve reliability.
  - Implement advanced features like traffic management, security, and observability.

- **Continuous Integration and Continuous Delivery (CI/CD):**
  - Automate the build, test, and deployment process.
  - Integrate with Kubernetes to streamline the deployment pipeline.

Post-migration best practices for Kubernetes

Some of the best practices they you should follow post-migration:

Image versioning and tagging

- **Semantic versioning: Use semantic versioning (MAJOR.MINOR.PATCH) to clearly identify changes in images.**
- **Image tagging:**
  - **Latest:** Points to the most recent image. Avoid using the latest tag images in production environments.
  - **Version tags:** Specific versions, e.g., v1.0.0.
  - **Environment tags:** Environment-specific tags, e.g., dev, staging, prod.
  - **Commit SHA tags:** Link images to specific Git commits for traceability.

Immutable infrastructure

- **Deploy new versions:** Instead of modifying existing deployments, deploy new versions with updated configurations.
- **Configuration management:** Use tools like Helm or Kustomize to manage configurations and updates.

Monitoring and optimization

- **Kubernetes Metrics server:** Collect metrics about cluster resources, nodes, pods, and containers.
- **Prometheus:** Monitor custom metrics and alerts.
- **Grafana:** Visualize metrics and create custom dashboards.
- **Alertmanager:** Set up alerts for critical issues, such as pod failures, resource exhaustion, and application errors.
- **Resource utilization:** Monitor CPU, memory, and disk usage to identify optimization opportunities.
- **Performance profiling:** Use tools like kubectl top pod and kubectl describe pod to analyze pod performance.
- **Rightsizing:** Adjust resource requests and limits to optimize resource allocation.
- **Autoscaling:** Use Horizontal Pod Autoscaler (HPA) to automatically scale applications based on load.

Security best practices

- **Image scanning:** Regularly scan container images for vulnerabilities.
- **Network policies:** Implement network policies to control traffic flow between pods.
- **Secrets management:** Use Kubernetes Secrets or external secrets management tools.
- **RBAC:** Enforce role-based access control to limit user permissions.
- **Regular security audits:** Conduct regular security assessments to identify and address vulnerabilities.

Continuous improvement

- **Stay updated:** Keep up-to-date with Kubernetes releases and security patches.
- **Continuous Integration/Continuous Delivery (CI/CD):** Automate the deployment pipeline.
- **A/B testing:** Experiment with different configurations and deployments.
- **Feedback loops:** Gather feedback from users and operations teams to improve future deployments.

Naviteq

Conclusion

Migrating from virtual machines to a containerized Kubernetes environment is a complex and transformative journey that requires careful planning, technical expertise, and strategic execution. While the benefits of containerization are substantial i.e improved scalability, efficiency, and deployment speed, the process is not without its fair share of challenges. Organizations often underestimate the intricacies involved in container migration, from redesigning application architectures to managing stateful services and ensuring secure, performant deployments. Given these complexities, partnering with experienced technology consultants like that from Naviteq can be a game-changer. Our services like Kubernetes Management, Monitoring and Logging Management, Infrastructure as a Code and CI/CD setup can help you in setting up a robust DevOps infrastructure. Our experts possess specialized knowledge in cloud-native technologies, enabling us to accelerate your migration process. This allows your team to concentrate solely on product development, freeing them from the burden of dealing with the intricate details of the migration itself.

Additional resources

- https://cloud.google.com/migrate/containers/docs/getting-started
- https://cloud.ibm.com/docs/solution-tutorials?topic=solution-tutorials-vm-to-containers-and-kubernetes
- https://www.youtube.com/watch?v=MuX3m149FpI
- https://kubernetes.io/docs/concepts/overview/

Ready to get started?

Contact Naviteq today to learn how our experts can help you containerize your applications with Kubernetes in an efficient and scalable manner.