Supporting Containers

After completing this episode, you should be able to:

- Discuss networking, storage, and registries typically found in container environments

Description: In this episode, you will learn about some of the different options when it comes to supporting container environments. This video includes information on port mapping, persistent volumes, ephemeral storage, and image registries.

Supporting Containers

Container networking involves establishing communication between containers running on the same host or across multiple hosts. One essential aspect of container networking is port mapping, which allows containers to expose and access services running inside them or in other containers.

Port mapping works by mapping a port on the host system to a port inside the container. This allows external traffic to reach the containerized service by accessing the corresponding port on the host. Conversely, it enables services inside containers to communicate with each other or with services running on the host or other containers.

For example, if a web server container exposes port 80 internally for HTTP traffic, you can map port 80 of the container to port 8080 on the host. Any requests made to port 8080 on the host will be forwarded to port 80 inside the container, allowing external users to access the web server.

Port mapping is typically configured during container creation using containerization platforms like Docker or Kubernetes. In Docker, you can specify port mappings using the `-p` or `--publish` flag when running a container. In Kubernetes, you define port mappings in the pod or service configurations.

Port mapping plays a vital role in container networking by enabling containers to expose services to the outside world and facilitating communication between containers and external systems. It allows for flexible and secure deployment of containerized applications while ensuring seamless connectivity and interoperability in distributed environments.

In a typical cloud infrastructure, container storage involves managing data persistence and storage for containerized applications. This includes two main components: persistent volumes and ephemeral storage.

Persistent Volumes: Persistent volumes are storage volumes that exist independently of containers and can be dynamically provisioned and attached to containers as needed. They provide a way to store and persist data beyond the lifecycle of individual containers. Persistent volumes are commonly used for storing application data, databases, or shared files that need to be retained even if the container is restarted or recreated. Cloud providers offer various options for persistent volumes, including block storage (e.g., AWS EBS, Azure Disk), file storage (e.g., AWS EFS, Azure Files), or object storage (e.g., AWS S3, Azure Blob Storage). Kubernetes, for example, allows administrators to define persistent volume claims (PVCs) that specify storage requirements, and the underlying infrastructure dynamically provisions and attaches the appropriate persistent volume to the container.

Ephemeral Storage: Ephemeral storage refers to temporary storage that is directly associated with the container instance and exists only for the duration of the container's lifecycle. Ephemeral storage is typically used for storing transient data, such as application logs, temporary files, or cache data, that does not need to be preserved beyond the container's lifetime. Ephemeral storage is usually provided as part of the container runtime environment and is stored on the underlying host's filesystem. In cloud environments, ephemeral storage may be backed by local SSDs or instance storage, offering fast access and low latency.

Image repositories (or registries) play a central role in a container-based infrastructure by serving as a centralized location for storing, distributing, and managing container images. These repositories store the immutable images that contain everything needed to run a containerized application, including the application code, runtime environment, dependencies, and configurations. Here's how image repositories are used in a container-based infrastructure:

Image Distribution: Container images are typically built and stored in a registry or repository, such as Docker Hub, Google Container Registry, or Amazon Elastic Container Registry. These repositories provide a secure and scalable platform for storing and distributing container images globally. Developers can push newly built images to the repository, making them available for deployment across different environments.

Version Control: Image repositories support versioning, allowing developers to track changes and updates to container images over time. Each image version is tagged with a unique identifier, such as a version number or a git commit hash, enabling precise control and management of image versions. Versioning ensures consistency and reproducibility in deployments, as developers can deploy specific versions of container images with confidence.

Image Lifecycle Management: Image repositories facilitate the management of the container image lifecycle, including image creation, storage, deletion, and garbage collection. Administrators can define policies and rules for image retention and cleanup, ensuring efficient use of storage resources and compliance with organizational requirements. Automated tools and workflows can be integrated with the repository to streamline image management tasks and enforce best practices.

Access Control and Security: Image repositories support access control mechanisms to regulate who can push, pull, or modify images stored in the repository. Role-based access control (RBAC), authentication, and authorization mechanisms help enforce security policies and prevent unauthorized access or tampering with images. Additionally, image repositories may offer scanning and vulnerability

detection features to identify and mitigate security risks in container images before deployment.

Integration with Orchestration Platforms: Container orchestration platforms like Kubernetes or Docker Swarm seamlessly integrate with image repositories to deploy and manage containerized applications at scale. Orchestration platforms pull container images from the repository based on deployment specifications, ensuring consistent and reliable deployments across clusters of nodes. Continuous integration and continuous deployment (CI/CD) pipelines can also leverage image repositories to automate the build, test, and deployment of containerized applications.

Additional resources

- Container Networking: https://www.vmware.com/topics/glossary/content/container-networking.html (https://www.vmware.com/topics/glossary/content/container-networking.html)