# Docker and Kubernetes Concepts Summary

## Key Components of a Dockerfile

A Dockerfile is a script that contains instructions to build a Docker image. Key components include:  
- FROM: Specifies the base image.  
- RUN: Executes commands during image build (e.g., installing packages).  
- COPY / ADD: Copies files from host to image.  
- WORKDIR: Sets the working directory inside the container.  
- CMD: Provides default command to run when the container starts.  
- ENTRYPOINT: Configures a container to run as an executable.  
- ENV: Sets environment variables.  
- ARG: Defines build-time variables.  
- EXPOSE: Documents the port the container listens on.  
- VOLUME: Creates mount points for persistent data.  
- LABEL: Adds metadata to the image.

## Difference Between CMD and ENTRYPOINT

CMD provides a default command to run when the container starts, and it can be overridden by command-line arguments.  
ENTRYPOINT sets the main command to run and is less flexible unless combined with CMD.  
Example:  
CMD ["npm", "start"]  
ENTRYPOINT ["python", "app.py"]

## Difference Between ARG and ENV in Dockerfile

ARG is used for build-time variables and is not preserved in the final image.  
ENV sets environment variables that are available during build and runtime.  
Example:  
ARG VERSION=1.0  
ENV VERSION=1.0

## How to Create and Remove a Docker Image

To create a Docker image:  
docker build -t my-image-name .  
  
To remove a Docker image:  
docker rmi my-image-name

## How to Expose a Running Container’s Ports

In Dockerfile:  
EXPOSE 8080  
  
When running the container:  
docker run -p 8080:8080 my-image-name

## Different Kubernetes Services

Kubernetes services expose pods to internal or external traffic. Types include:  
1. ClusterIP: Accessible only within the cluster.  
2. NodePort: Exposes the service on a static port on each node.  
3. LoadBalancer: Provisions an external load balancer (cloud environments).  
4. ExternalName: Maps service to a DNS name (no proxying).  
5. Headless Service: No cluster IP; used for direct pod access (e.g., StatefulSets).

**Jenkins Conversation Summary**

**Types of Pipelines in Jenkins**

1. Scripted Pipeline: Uses Groovy-based DSL, more flexible but complex. Defined using 'node' blocks.

2. Declarative Pipeline: Structured and readable syntax using 'pipeline' blocks. Easier to maintain.

**Setting Up a Daily Job Using Cron**

Use 'Build periodically' trigger in job configuration.

Example cron expression: 'H 0 \* \* \*' runs the job daily at midnight.

Jenkins uses 'H' to distribute load evenly.

**Running a Second Job While One is Running**

Enable 'Execute concurrent builds if necessary' in job settings.

Use 'build job: ..., wait: false' in pipeline to trigger another job asynchronously.

Use post-build actions or Parameterized Trigger Plugin for more control.

**Storing Credentials in Jenkins Pipeline**

Add credentials in Jenkins under 'Manage Jenkins' > 'Credentials'.

Use 'environment' block or 'withCredentials' in pipeline to access secrets.

Supports secret text, username/password, SSH keys, etc.

**Debugging a Failed Jenkins Job**

Check console output for errors and stack traces.

Review Jenkinsfile or job configuration for syntax and logic issues.

Validate environment, credentials, dependencies, and plugin compatibility.

Use debug flags and Jenkins system logs for deeper inspection.

**Triggering a Job Based on Memory Usage**

Use external monitoring script or tool to check memory usage.

Trigger Jenkins job via CLI or REST API when threshold is exceeded.

Example: Python script using psutil and requests to monitor and trigger.

**Integrating Autoscaling with Jenkins**

Use Kubernetes Plugin to run agents as ephemeral pods.

Use EC2 Plugin with Auto Scaling Groups for AWS.

Docker Plugin for container-based agents.

Azure VM Agents and GCP Compute Engine Plugin for cloud autoscaling.

Jenkins Concepts Summary

1. Types of Pipelines in Jenkins

Scripted Pipeline

Scripted Pipelines use Groovy-based DSL and are written inside a 'node' block. They offer more flexibility and control but are more complex.

Example:

node {

stage('Build') {

echo 'Building...'

}

stage('Test') {

echo 'Testing...'

}

stage('Deploy') {

echo 'Deploying...'

}

}

Declarative Pipeline

Declarative Pipelines use a structured syntax with a 'pipeline' block. They are easier to read and maintain, making them ideal for most CI/CD workflows.

Example:

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building...'

}

}

stage('Test') {

steps {

echo 'Testing...'

}

}

stage('Deploy') {

steps {

echo 'Deploying...'

}

}

}

}

2. Setting Up a Daily Job Using Cron Expression

To schedule a daily job in Jenkins, go to the job configuration, enable 'Build periodically', and enter a cron expression like 'H 0 \* \* \*' to run the job at midnight every day.

Common Daily Cron Expressions:

Midnight: H 0 \* \* \*

6 AM: H 6 \* \* \*

Noon: H 12 \* \* \*

6 PM: H 18 \* \* \*

3. Triggering a Second Job While One is Running

You can trigger another job from a running job using the 'build' step in a pipeline or post-build actions.

Example using pipeline:

pipeline {

agent any

stages {

stage('Run Job A') {

steps {

echo 'Running Job A...'

}

}

stage('Trigger Job B') {

steps {

build job: 'JobB', wait: false

}

}

}

}

4. Storing Credentials in Jenkins Pipeline

Use the Credentials Plugin to securely store and access secrets in Jenkins pipelines.

Example for Secret Text:

pipeline {

agent any

environment {

MY\_SECRET = credentials('my-secret-token')

}

stages {

stage('Use Secret') {

steps {

sh 'echo "Using secret: $MY\_SECRET"'

}

}

}

}

Example for Username and Password:

withCredentials([usernamePassword(credentialsId: 'my-creds-id', usernameVariable: 'USER', passwordVariable: 'PASS')]) {

sh 'echo "Logging in with $USER and $PASS"'

}

5. Debugging a Failed Jenkins Job

Steps to debug a failed job include checking console output, reviewing job configuration, validating environment and credentials, inspecting dependencies, and using debug flags.

6. Triggering a Job Based on Memory Usage

Use a monitoring script or external tools to check memory usage and trigger Jenkins jobs via CLI or REST API.

Example Python Snippet:

import psutil

import requests

memory = psutil.virtual\_memory()

threshold = 80 # percent

if memory.percent > threshold:

requests.post(

'http://your-jenkins-url/job/your-job-name/build',

auth=('youruser', 'yourtoken')

)

7. Integrating Autoscaling with Jenkins

Jenkins can integrate with cloud platforms for autoscaling build agents. Common setups include Kubernetes, AWS EC2, Docker, Azure VM Scale Sets, and GCP.

Key Plugins for Autoscaling:

- Kubernetes Plugin

- EC2 Plugin

- Docker Plugin

- Azure VM Agents

- GCP Compute Engine Plugin

**Kubernetes and AWS Q&A Summary**

**What are Node Affinity & Non-Affinity?**

Node Affinity attracts pods to specific nodes based on labels. It includes required (hard) and preferred (soft) rules. Non-Affinity (Anti-Affinity) avoids placing pods on certain nodes or near other pods. It includes Pod Anti-Affinity and Node Anti-Affinity.

**Can you explain with examples?**

Yes. Examples include YAML configurations for required and preferred Node Affinity, and Pod Anti-Affinity to avoid placing similar pods on the same node.

**Difference between Pod Affinity & Pod Anti-Affinity.**

Pod Affinity co-locates pods with others based on labels, useful for communication. Pod Anti-Affinity separates pods for high availability. Both use topology keys like hostname.

**What are Hard vs. Soft conditions in Pod Affinity/Anti-Affinity.**

Hard conditions (requiredDuringSchedulingIgnoredDuringExecution) are mandatory for scheduling. Soft conditions (preferredDuringSchedulingIgnoredDuringExecution) are best-effort and allow scheduling even if not met.

**What are common scenarios for each condition?**

Hard: Security compliance, hardware needs, data locality, isolation. Soft: Performance optimization, load distribution, cost efficiency, co-location.

**Explain HPA (Horizontal Pod Autoscaler) and VPA (Vertical Pod Autoscaler).**

HPA scales the number of pods based on metrics like CPU. VPA adjusts CPU/memory requests per pod. HPA is for stateless apps; VPA is for resource-intensive apps.

**What conditions can be set in HPA?**

Conditions include resource metrics (CPU, memory), custom metrics (e.g., RPS), external metrics (e.g., queue length), and behavior settings like stabilization windows.

**What is Ingress in Kubernetes, give me a real time example**

Ingress manages external access to services. Example: routing '/' to frontend-service and '/api' to api-service using a single domain like example.com.

**How would you debug a domain that isn’t reaching a Kubernetes service?**

Check DNS resolution, Ingress controller, Ingress resource, service configuration, pod health, internal access, network policies, TLS, and cloud firewall/load balancer.

**If a service is returning a 500 error, how would you troubleshoot it?**

Check pod logs, container health, application code, service routing, test locally, inspect Ingress, monitor resources, check dependencies, and enable debug logging.

**Explain Rolling Updates and restarts in Kubernetes.**

Rolling Updates gradually replace pods with new ones. Restarts can be manual, crash recovery, probe-triggered, or rolling restarts using kubectl.

**When restarting using kubectl, will there be downtime?**

Usually no, if multiple replicas and readiness probes are configured. Downtime may occur with single replicas or misconfigured probes.

**How do you access an S3 bucket from Account A in Account B**

Use cross-account IAM roles and bucket policies. Account A allows Account B's role in the bucket policy. Account B assumes the role or accesses directly if trusted.

**How do you connect an EC2 instance to an S3 bucket?**

Create an IAM role with S3 permissions, attach it to the EC2 instance, and use AWS CLI or SDK to access the bucket without credentials.