**CSE5006 Essay Exam Questions Examples**

**Some very short coding (18)**

**1) SVN - Git**

**Q: How do you merge a branch into the current branch in Git, and resolve conflicts?**

**A:**

git merge feature-branch

# Resolve conflicts manually in the conflicted files

git add <resolved-files>

git commit

**Q: How do you clone a Git repository?**

**A:**

git clone https://github.com/user/repository.git

**2) Container - Docker**

**Q: How do you create a multi-stage Dockerfile to optimize the image size?**

**A:**

# Build stage

FROM node:16 AS build

WORKDIR /app

COPY . .

RUN npm install

# Production stage

FROM node:16-slim

WORKDIR /app

COPY --from=build /app .

RUN npm run build

CMD ["npm", "start"]

**3) HTML & JavaScript**

**Q: How do you implement form validation in JavaScript to check if a field is empty before submission?**

<form id="myForm">

<input type="text" id="name" placeholder="Enter your name">

<button type="submit">Submit</button>

</form>

<script>

document.getElementById("myForm").addEventListener("submit", function(event) {

const name = document.getElementById("name").value;

if (!name) {

event.preventDefault();

alert("Name is required!");

}

});

</script>

**4) React**

**Q: How do you implement controlled components in React for form input handling?**

import { useState } from 'react';

const MyForm = () => {

const [inputValue, setInputValue] = useState('');

const handleChange = (e) => {

setInputValue(e.target.value);

};

const handleSubmit = (e) => {

e.preventDefault();

alert(inputValue);

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

value={inputValue}

onChange={handleChange}

placeholder="Enter something"

/>

<button type="submit">Submit</button>

</form>

);

};

**5) ORM - Sequelize**

**Q: How do you perform a complex query with Sequelize to join two tables and filter results?**

User.findAll({

include: [{

model: Post,

where: { title: { [Sequelize.Op.like]: '%Hello%' } },

required: true

}],

where: { age: { [Sequelize.Op.gt]: 18 } }

})

.then(users => console.log(users));

**6) How do you implement a custom middleware in Express to log request details?**

const express = require('express');

const app = express();

const requestLogger = (req, res, next) => {

console.log(`${req.method} ${req.url}`);

next();

};

app.use(requestLogger);

app.get('/', (req, res) => {

res.send('Hello World');

});

app.listen(3000, () => {

console.log('Server running on port 3000');

});

**7) Proxy and Cloud Network**

**Q: How do you set up a reverse proxy using Nginx to forward traffic to a backend service?**

server {

listen 80;

location / {

proxy\_pass http://localhost:3000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

**8) AWS Cloud Fundamentals**

**Q: How do you create an IAM policy to allow only S3 bucket access from a specific IP address range?**

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:\*",

"Resource": "arn:aws:s3:::my-bucket/\*",

"Condition": {

"IpAddress": {

"aws:SourceIp": "203.0.113.0/24"

}

}

}

]

}

**9) AWS Cloud Storage**

**Q: How do you configure an S3 bucket to trigger an AWS Lambda function when a new file is uploaded?**

aws lambda create-event-source-mapping --function-name MyLambdaFunction --event-source-arn arn:aws:s3:::my-bucket --event-source-type s3

**10) AWS Cloud Containers**

**Q: How do you define an ECS task definition with environment variables in AWS CLI?**

aws ecs register-task-definition --family my-task \

--container-definitions '[

{

"name": "my-container",

"image": "my-image",

"environment": [

{"name": "ENV\_VAR", "value": "value"}

]

}

]'

**11) CI/CD in AWS**

**Q: How do you create an AWS CodeBuild project with environment variables?**

aws codebuild create-project --name my-build-project \

--source type=CODEPIPELINE,buildspec=buildspec.yml \

--artifacts type=CODEPIPELINE \

--environment type=LINUX\_CONTAINER,image=aws/codebuild/standard:4.0,environmentVariables=[{name=MY\_VAR,value=my\_value}]

**12) SVN - Git**

**Q: How do you handle a conflict during a merge in Git?**

git merge feature-branch

# Git will mark conflicts in the affected files

# Open the files and resolve conflicts manually

git add <resolved-files>

git commit

**13) Container - Docker**

**Q: How do you create a Docker Compose file to run a web app with a database container?**

version: '3'

services:

web:

image: nginx

ports:

- "8080:80"

db:

image: postgres

environment:

POSTGRES\_PASSWORD: example

14) **HTML & JavaScript**

**Q: How do you create a dynamic table in HTML using JavaScript?**

<table id="myTable"></table>

<script>

const data = [['Name', 'Age'], ['John', 30], ['Jane', 25]];

const table = document.getElementById('myTable');

data.forEach(row => {

const tr = document.createElement('tr');

row.forEach(cell => {

const td = document.createElement('td');

td.textContent = cell;

tr.appendChild(td);

});

table.appendChild(tr);

});

</script>

**15) React**

**Q: How do you implement a custom hook in React to fetch data from an API?**

import { useState, useEffect } from 'react';

const useFetch = (url) => {

const [data, setData] = useState(null);

const [loading, setLoading] = useState(true);

useEffect(() => {

fetch(url)

.then(res => res.json())

.then(data => {

setData(data);

setLoading(false);

});

}, [url]);

return { data, loading };

};

const MyComponent = () => {

const { data, loading } = useFetch('https://api.example.com/data');

if (loading) return <div>Loading...</div>;

return <div>{JSON.stringify(data)}</div>;

};

**16) ORM - Sequelize**

**Q: How do you perform a transaction with Sequelize to ensure data consistency?**

const sequelize = require('sequelize');

const { User, Order } = require('./models');

sequelize.transaction(async (t) => {

const user = await User.create({ name: 'John Doe' }, { transaction: t });

const order = await Order.create({ userId: user.id, amount: 100 }, { transaction: t });

await t.commit();

}).catch((error) => {

console.error(error);

});

**17) REST APIs**

**Q: How do you implement pagination in a REST API using Express?**

app.get('/items', async (req, res) => {

const page = parseInt(req.query.page) || 1;

const pageSize = parseInt(req.query.pageSize) || 10;

const offset = (page - 1) \* pageSize;

const items = await Item.findAll({ offset, limit: pageSize });

res.json(items);

});

**18) AWS Cloud Fundamentals**

**Q: How do you create an S3 bucket and enable versioning using AWS CLI?**

aws s3api create-bucket --bucket my-bucket --region us-west-2

aws s3api put-bucket-versioning --bucket my-bucket --versioning-configuration Status=Enabled

**19) AWS Cloud Storage**

**Q: How do you configure S3 bucket lifecycle rules to archive objects to Glacier after 30 days?**

{

"Rules": [

{

"ID": "ArchiveToGlacier",

"Status": "Enabled",

"Filter": {},

"Transitions": [

{

"Days": 30,

"StorageClass": "GLACIER"

}

]

}

]

}

**20) AWS Cloud Containers**

**Q: How do you define a simple ECS service that runs a container in a Fargate cluster?**

aws ecs create-service \

--cluster my-cluster \

--service-name my-service \

--task-definition my-task \

--desired-count 1 \

--launch-type FARGATE

**Coding analysis (18)**

**1) SVN - Git**

**Q: What is the difference between SVN and Git?**

**A:**

* **SVN (Subversion)** is a centralized version control system. It relies on a central server to store the version history, and developers check out files from that central server.
* **Git** is a distributed version control system. Every user has their own local repository, which allows them to work offline and commit changes locally before pushing them to a shared repository.

**2) Container - Docker**

**Q: What is Docker?**

**A:**  
Docker is a platform that allows developers to automate the deployment of applications inside lightweight, portable containers. These containers are isolated from the system, making it easier to run applications consistently across different environments.

**3) HTML & JavaScript**

**Q: How do you create a button in HTML and add a JavaScript function to it?**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Button Example</title>

</head>

<body>

<button onclick="displayMessage()">Click Me</button>

<script>

function displayMessage() {

alert("Hello, World!");

}

</script>

</body>

</html>

**4) React**

**Q: What is a functional component in React?**

**A:**  
A **functional component** in React is a simple JavaScript function that accepts props as an argument and returns JSX to be rendered on the screen. They do not have internal state (unless using hooks).

import React from 'react';

const Greeting = (props) => {

return <h1>Hello, {props.name}!</h1>;

}

export default Greeting;

**5) ORM - Sequelize**

**Q: What is Sequelize in Node.js?**

**A:**  
Sequelize is an Object-Relational Mapping (ORM) library for Node.js that allows developers to interact with relational databases like MySQL, PostgreSQL, and SQLite using JavaScript, instead of writing raw SQL queries.

**CODE**

const { Sequelize, DataTypes } = require('sequelize');

const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', {

name: {

type: DataTypes.STRING,

allowNull: false

},

});

(async () => {

await sequelize.sync();

const user = await User.create({ name: 'John Doe' });

console.log(user.name); // John Doe

})();

**6) REST APIs**

**Q: What is a REST API?**

**A:**  
A **REST (Representational State Transfer)** API is an architectural style for designing networked applications. It uses HTTP requests to perform CRUD operations (Create, Read, Update, Delete) and is stateless, meaning each request from a client contains all the information needed to process it.

**Example of a simple GET request using Fetch in JavaScript:**

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => console.log(data));

**7) Proxy and Cloud Network**

**Q: What is a proxy server?**

**A:**  
A **proxy server** acts as an intermediary between a client and a destination server. It can provide benefits such as security, load balancing, and caching by forwarding requests from clients to servers.

**8) AWS Cloud Fundamentals**

**Q: What is AWS?**

**A:**  
AWS (Amazon Web Services) is a cloud computing platform that offers a variety of services like computing power, storage, and databases over the internet. AWS allows businesses to scale their operations without the need for physical hardware.

**9) AWS Cloud Storage**

**Q: What is Amazon S3 in AWS?**

**A:**  
Amazon S3 (Simple Storage Service) is a scalable cloud storage service provided by AWS. It allows users to store and retrieve any amount of data, including backups, media files, and more, using a web interface.

aws s3 cp myfile.txt s3://my-bucket-name/

**10) AWS Cloud Containers**

**Q: What is Amazon ECS (Elastic Container Service)?**

**A:**  
Amazon ECS is a container management service that supports Docker containers. It helps you easily run applications in containers on a scalable infrastructure. ECS handles the orchestration and scaling of the containers.

**11) CI/CD in AWS**

**Q: What is CI/CD in AWS?**

**A:**  
CI/CD (Continuous Integration and Continuous Delivery) in AWS refers to the practice of automatically integrating code changes and deploying them to production without manual intervention.

**Example with AWS CodePipeline:**

* **CI (Continuous Integration):** Code changes are automatically tested and merged into the main branch.
* **CD (Continuous Delivery):** Changes are automatically deployed to staging or production environments using AWS CodeDeploy or Elastic Beanstalk.

**general knowledge (14)**

**1) SVN - Git**

**Q: Explain the concept of distributed version control in Git and discuss its advantages over centralized systems like SVN.**  
**A:**  
Git is a **distributed version control system**, meaning each developer has a full copy of the repository, including the entire history. Unlike SVN, which relies on a central server, Git allows developers to work offline, make local commits, and push changes to a remote repository when ready.  
**Advantages:**

* **Offline Work**: Developers can commit changes and view history without an internet connection.
* **Speed**: Operations like commits and diffs are faster because they’re done locally.
* **Better Collaboration**: Multiple developers can work on separate branches independently and merge their work easily.

**Q: Explain the concept of rebasing in Git. What are the advantages of using rebasing over merging when maintaining a feature branch?**  
**A:**  
Rebasing re-applies commits from one branch on top of another branch’s tip, creating a linear history.  
**Advantages**:

* **Clean Project History**: Simplifies commit logs, avoiding merge commits.
* **Easier Review**: A linear history is easier to review and understand.

**2) Container - Docker**

**Q: Explain the concept of containerization in Docker and discuss its benefits compared to traditional virtual machines.**  
**A:**  
Containerization involves packaging an application and its dependencies together into a **container**. Docker makes this possible by providing a consistent environment for development and deployment.  
**Advantages:**

* **Lightweight**: Containers share the host OS kernel, making them more efficient than VMs.
* **Portability**: Containers can run anywhere Docker is installed, ensuring consistent behaviour across environments.
* **Fast Startup**: Containers start almost instantly compared to VMs, which boot up an entire OS.

**Q: Discuss how Docker’s layered file system contributes to the efficiency of building and managing containers.**  
**A:**  
Docker images are built in layers, where each layer represents an instruction in a Dockerfile.  
**Advantages**:

* **Reusability**: Layers that don’t change are cached, speeding up the build process.
* **Smaller Updates**: Only modified layers need to be rebuilt, reducing image size and build time.

**3) HTML & JavaScript**

**Q: Explain the concept of the Document Object Model (DOM) in JavaScript and its role in web development.**  
**A:**  
The **DOM** is a programming interface for HTML and XML documents. It represents the page as a tree structure, allowing JavaScript to access and manipulate the content, structure, and styles dynamically.  
**Advantages:**

* **Interactive Webpages**: Developers can change elements, update content, and respond to user actions without reloading the page.
* **Event Handling**: The DOM allows JavaScript to respond to user events like clicks and keypresses.
* **Dynamic Content**: Pages can be modified after they’re loaded, enabling interactive web applications.

**4) React**

**Q: Discuss how the virtual DOM in React optimizes the rendering process and enhances application performance.**  
**A:**  
The **virtual DOM** is an in-memory representation of the real DOM. React updates the virtual DOM first, calculates changes, and then applies minimal updates to the real DOM.  
**Advantages**:

* **Efficient Rendering**: Reduces the number of direct manipulations of the DOM, which are slow.
* **Batch Updates**: React groups updates to minimize reflows and repaints.

**5) ORM - Sequelize**

**Q: Explain the concept of eager loading in Sequelize. What are the advantages of using eager loading when querying data?**  
**A:**  
**Eager loading** retrieves associated data models in a single query, avoiding multiple database calls.  
**Advantages**:

* **Reduced Latency**: Fewer database calls lead to faster query execution.
* **Simplified Code**: Data relationships are fetched together, making code easier to read.

**6) REST APIs**

**Q: Discuss how RESTful APIs achieve statelessness and explain the advantages of this approach in web development.**  
**A:**  
REST APIs are stateless, meaning each request from the client contains all the information needed for the server to process it.  
**Advantages**:

* **Scalability**: Servers can handle multiple requests independently, improving load distribution.
* **Simplified Architecture**: No need to store session state on the server

**7) Proxy and Cloud Network**

**Q: Explain the concept of reverse proxy servers. What are the advantages of using a reverse proxy in cloud-based applications?**  
**A:**  
A **reverse proxy** sits in front of web servers, forwarding client requests to the appropriate server.  
**Advantages**:

* **Load Balancing**: Distributes traffic evenly across servers.
* **Security**: Can hide the identity and structure of the backend server, adding an extra layer of security.

**8) AWS Cloud Fundamentals**

**Q: Explain how AWS Regions and Availability Zones contribute to high availability and fault tolerance.**  
**A:**  
AWS Regions are separate geographic areas, and each Region contains multiple **Availability Zones** (AZs). AZs are isolated data centers that allow resources to be distributed across multiple locations.  
**Advantages**:

* **High Availability**: If one AZ fails, others can maintain the workload.
* **Fault Tolerance**: Spreading applications across multiple AZs reduces the risk of downtime.

**8) AWS Cloud Fundamentals**

**Q: Discuss how the shared responsibility model works in AWS. What are the advantages of understanding this model for cloud users?**  
**A:**  
The **shared responsibility model** divides security tasks between AWS and its users. AWS manages the security *of* the cloud (physical infrastructure, networking), while users manage the security *in* the cloud (data encryption, application-level controls).  
**Advantages**:

**Clarity in Responsibilities**: Users know what they are accountable for, reducing potential security gaps.

**Improved Security**: Both parties focus on their respective areas, ensuring comprehensive protection.

**9) AWS Cloud Storage**

**Q: Discuss how Amazon S3’s versioning feature works and what advantages it provides.**  
**A:**  
S3 **versioning** maintains multiple versions of an object, allowing you to retrieve, restore, or delete older versions.  
**Advantages**:

* **Data Protection**: Safeguards against accidental deletions or overwrites.
* **Backup and Recovery**: Easily revert to an older version if needed.

**Q: Explain the concept of Amazon S3’s storage classes and discuss their advantages for cost optimization.**  
**A:**  
**Amazon S3** offers different storage classes (e.g., S3 Standard, S3 Intelligent-Tiering, S3 Glacier) to meet varying access and cost requirements.  
**Advantages**:

* **Cost Efficiency**: Data that is infrequently accessed can be moved to cheaper storage classes, reducing overall costs.
* **Automatic Tiering**: The Intelligent-Tiering class shifts data automatically to the most cost-effective tier based on access patterns.

**10) AWS Cloud Containers**

**Q: Explain how Amazon ECS and EKS differ. What are the advantages of using one over the other?**  
**A:**  
**Amazon ECS** is a container orchestration service that is fully managed and integrates tightly with AWS services. **Amazon EKS** is a managed Kubernetes service.  
**Advantages**:

* **ECS**: Easier to use, simpler for teams already familiar with AWS.
* **EKS**: Supports Kubernetes’ full ecosystem, ideal for teams that need flexibility and multi-cloud capabilities.

**Q: Discuss how Amazon ECS (Elastic Container Service) simplifies container management. What are the advantages of using ECS compared to managing containers manually?**  
**A:**  
**Amazon ECS** is a managed service that orchestrates container deployment, scaling, and management.  
**Advantages**:

* **Simplified Management**: Removes the need to manage infrastructure manually, allowing focus on building applications.
* **Integration with AWS Services**: Seamless integration with AWS services like IAM, CloudWatch, and ELB for enhanced security and monitoring.
* **Scalability**: Automatically scales services up or down based on demand.

**11) CI/CD in AWS**

**Q: Explain how AWS CodePipeline automates the CI/CD process and discuss its advantages.**  
**A:**  
**AWS CodePipeline** automates the build, test, and deployment phases of your release process.  
**Advantages**:

* **Continuous Integration/Deployment**: Automates repetitive tasks, speeding up the release cycle.
* **Scalability**: Integrates with other AWS services and third-party tools, allowing for flexible pipelines.

**11) CI/CD in AWS**

**Q: Explain the concept of continuous integration and continuous deployment (CI/CD) pipelines in AWS. What are the advantages of implementing CI/CD using AWS tools like CodePipeline and CodeBuild?**  
**A:**  
**CI/CD** pipelines automate the process of integrating code changes, building, testing, and deploying applications. AWS provides services like **CodePipeline** for orchestrating the process and **CodeBuild** for compiling code and running tests.  
**Advantages**:

* **Automation**: Reduces manual effort, accelerating the development cycle.
* **Consistency**: Ensures code is tested and deployed in a consistent manner, reducing errors.
* **Scalability**: Handles large projects and multiple developers efficiently, scaling as needed.