DevOps
AIDE501 – M2 – 3 Credits
2026/2027

Course Contents

- **Chapter 1: Cloud Computing**
- **Chapter 2: Containerization**
- **Chapter 3: Shell Scripting**
- **Chapter 4: DevOps**
- **Chapter 5: Orchestration**
- **Chapter 6: GitLab CI/CD**

Chapter 3:

Shell Scripting

Chap.3: Shell Scripting

- Basics of Shell Scripting
- Advanced Scripting Techniques
- YAML, JSON

YAML and JSON

YAML and JSON are essential formats for configuring modern software infrastructure and automating deployment processes in the context of DevOps, Docker, Kubernetes, and CI/CD.

Modern applications consist of multiple components that need to exchange information and configurations:

- A frontend application communicates with a backend through APIs.
- Docker Compose uses YAML to define multi-container applications.
- Kubernetes uses YAML manifests to describe infrastructure.
- CI/CD systems use YAML to define automated workflows.
- REST APIs frequently exchange data in JSON format.

YAML and JSON

DevOps Example: Docker, Kubernetes, Terraform & Jenkins

1. Dockerfile

```
> FROM node:18
> WORKDIR /app
> COPY package*.json ./
> RUN npm install
> COPY ...
> EXPOSE 3000
> CMD ["npm", 'start"]
```

2. Docker-Compose

```
version: "3.8"
services:
  app:
    build. .
   ports:
     - "3000:3000"
   depends_on:
     - mongo
 mongo:
  image: mongo
  ports:
   - "27017:27017"
```

3. Pod.yaml

```
apiVersion: v1
kind: Pod
 metadata:
   name: node-app-pod
 spec:
  replicas: 1
     containers:
      - name:  node-container
        image: your-dockerhub-username/
               node-app:latest
        ports:
    }
```

4. Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-app-deployment
spec:
   replicas: 2
   selector:
      matchLabels:
         app: node-app
      template:
          metadata:
          labels:
      app: node-app
       spec: {
          containers:
       - containerPort: 3000
    }
```

5. Terraform (main.tf)

```
provider "aws" {
   region: "ap-south-1"

resource "aws_instance" "devops_server" {
   ami = "ami-0c655b159ebfafe1f0"
   instance_type: "t2.micro"
   tags = {
       Name = "DevOps-Server"
    }
}
```

6. Jenkins Pipeline

```
pipeline {
   agent any
 stages {
    stage( Checkuut Code)" {
     steps {
        git "https://github.com/
        your-repo/node-app.git"
     })
   },
   stage 'Build Docker Image' {
     steps {
        sh "docker build -t
        your-dockerhub-username/node-app..
     })
   },
   stage 'Push to DockerHub" {
     steps {
        sh "docker login -u
        your-username  -p -$ PPASS"
     })
   }
}
```



#Docker #Kubernetes #Terraform #Jenkins #DevOps #Cloud #CI/CD

Data serialization

Data serialization is the process of converting structured data into a format that can be:

- Stored in files.
- Transmitted over networks.
- Exchanged between applications.
- Read and processed by different programming languages.
- Used to configure software systems.

For example, consider a user represented by the following information:

- Name: John
- Age: 25
- Role: Developer

This information can be represented using JSON or YAML.

JSON (JavaScript Object Notation)

- It is a lightweight, text-based data interchange format used to exchange structured information between applications.
- Despite its name, JSON is language-independent and is supported by Java, Python, JavaScript, PHP, C#, and many other programming languages.
- JSON uses two main structures:
  - Objects: collections of key-value pairs.
  - Arrays: ordered collections of values.

```
{
  "name": "John",
  "age": 25,
  "isStudent": false,
  "skills": ["Java", "Python", "Docker"]
}
```

JSON (JavaScript Object Notation)

Data type	Description	Example
String	Text	`"John"`
Number	Integer or decimal	`25, 3.14`
Boolean	True or false	`true`
Null	Empty value	`null`
Object	Key-value collection	`{ "age": 25 }`
Array	Ordered collection	`[1, 2, 3]`

JSON (JavaScript Object Notation)

A JSON object is enclosed in curly braces `{}`.

Each property consists of:

- A key (string)
- A colon `:`
- A value
- A comma separating properties, except after the last property

```
{
  "server": {
    "hostname": "web-server",
    "ip": "192.168.1.10",
    "port": 8080
  }
}
```

JSON (JavaScript Object Notation)

Arrays are enclosed in square brackets `[]`.

```
{
  "students": [
    {
      "id": 1,
      "name": "Ali"
    },
    {
      "id": 2,
      "name": "Sara"
    }
  ]
}
```

JSON (JavaScript Object Notation)

Important JSON rules:

- Keys must be enclosed in double quotes.
- String values must use double quotes.
- Single quotes are not allowed.
- Comments are not supported in standard JSON.
- Trailing commas are not allowed.
- Object properties must be separated by commas.
- JSON does not support native date or function types.

YAML (YAML Ain't Markup Language)

YAML is a human-readable data serialization language.

It is designed to represent structured data in a simple and readable format.

YAML is widely used in:

- Docker Compose.
- Kubernetes.
- Ansible.
- GitHub Actions.
- GitLab CI/CD.
- Infrastructure as Code tools.

YAML (YAML Ain't Markup Language)

Unlike JSON, YAML uses indentation to represent hierarchy.

Example:

```
name: John
age: 25
isStudent: false
skills:
  - Java
  - Python
  - Docker
```

YAML (YAML Ain't Markup Language)

Notice:

- No curly braces are needed.
- No quotation marks are generally required around simple strings.
- Indentation defines the structure.
- Lists use hyphens (-).
- Key-value pairs use colons (:).

YAML (YAML Ain't Markup Language)

Data type	Example
String	`name: John`
Integer	`age: 25`
Float	`price: 19.99`
Boolean	`active: true`
Null	`value: null`
List	`ports: [80, 443]`
Object	Nested key-value pairs

YAML (YAML Ain't Markup Language)

YAML objects

```
server:
  hostname: web-server
  ip: 192.168.1.10
  port: 8080
```

The indentation indicates that `hostname`, `ip`, and `port` belong to the `server` object.

YAML (YAML Ain't Markup Language)

YAML lists

```
students:
  - id: 1
    name: Ali

  - id: 2
    name: Sara

technologies:
  - Docker
  - Kubernetes
  - Jenkins
  - Ansible
```

YAML (YAML Ain't Markup Language)

YAML allows complex nested data structures.

```
application:
  name: university-system
  version: 1.0

  database:
    host: localhost
    port: 5432
    username: admin

  services:
    - name: frontend
      port: 3000

    - name: backend
      port: 5000
```

YAML (YAML Ain't Markup Language)

Comments begin with `#`.

```
# Application configuration
application:
  name: my-app
  version: 1.0

  # Database configuration
  database:
    host: localhost
    port: 5432
```

YAML (YAML Ain't Markup Language)

YAML supports multiline strings using two important indicators:

Literal block (|): Preserves line breaks.

```
description: |
  This is a DevOps course.
  Students will learn Docker.
  Students will also learn Kubernetes.
```

Folded block (>): Folds most line breaks into spaces.

```
description: >
  This is a DevOps course
  about automation and
  continuous deployment.
```

YAML Environment variables

- Configuration files often reference environment variables.
- This allows sensitive or environment-specific configuration to be supplied externally.
- Security recommendation: Never hardcode passwords, API keys, or database credentials in configuration files committed to Git. Use environment-specific secret management, such as Docker secrets, Kubernetes Secrets with appropriate encryption and access controls, or a dedicated secrets manager.

```
services:
  backend:
    image: my-backend:latest
    environment:
      NODE_ENV: production
      PORT: "5000"
      DATABASE_URL: ${DATABASE_URL}
```

YAML Anchors and aliases

- To define reusable configuration sections. This reduces duplication in configuration files.

```
default_config: &default
  restart: always
  environment:
    NODE_ENV: production

frontend:
  <<: *default
  image: nginx:alpine

backend:
  <<: *default
  image: node:22-alpine
```

YAML Anchors and aliases

- Anchor (&): Place an ampersand followed by a name right before a scalar value, list, or mapping block to save it for reuse.
- Alias (*). Place an asterisk followed by the anchor name anywhere later in the document to paste that saved data.
- Merge Key (<<). Combine an alias with <<: inside a mapping to inherit all its key-value pairs while overriding or adding specific fields

YAML Anchors and aliases

```
common_image: &app_img ://example.com

services:
 web:
  image: *app_img
  ports: &default_ports
   - "80:80“
   - "443:443“
 api:
  image: *app_img
  ports: *default_ports
```

Multiple documents in one YAML file

- This is particularly useful in Kubernetes, where multiple resources can be defined in one manifest file.

```
---
name: frontend
port: 3000


---
name: backend
port: 5000


---
name: database
port: 3306
```

YAML in Docker Compose

Docker Compose uses YAML files to define and manage multi-container applications.

Example: `compose.yaml`

```
services: #Defines the containers

  frontend:
    image: nginx:alpine
    ports:
      - "8080:80"

  backend:
    image: node:22-alpine
    working_dir: /app
    volumes:   #Mounts a directory into the container
      - ./backend:/app
    command: node server.js #Specifies the command executed inside the container
    ports:
      - "5000:5000"
```

YAML in Kubernetes

- Kubernetes uses YAML manifests to describe the desired state of applications and infrastructure.
- For example, the following YAML file creates a Deployment with two replicas.

```
apiVersion: apps/v1    #Kubernetes API version
kind: Deployment       #Resource type

metadata:              #Resource identification
  name: nginx-deployment

spec:                  #Desired configuration
  replicas: 2

  selector:            #Identifies matching Pods
    matchLabels:
      app: nginx
```

YAML in Kubernetes

```
  template:
    metadata:
      labels:
        app: nginx

    spec:
      containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
```

Deploy the application:

```
kubectl apply -f deployment.yaml
```

Check the deployment:

```
kubectl get deployments
```

Check the Pods:

```
kubectl get pods
```

YAML in CI/CD pipelines

- CI/CD tools use YAML to define automated software development workflows.
- Example: GitHub Actions.
- File: .github/workflows/build.yml

```
name: Node.js CI          #Workflow name

on:                       #Events that trigger the workflow
  push:
    branches:
      - main

jobs:                     #Tasks to execute
  build:

    runs-on: ubuntu-latest   #Execution environment
```

YAML in CI/CD pipelines

```
steps:    #Individual actions and commands
  - name: Checkout source code
    uses: actions/checkout@v4 #Reuses an existing GitHub Action

  - name: Install Node.js
    uses: actions/setup-node@v4
    with:
      node-version: 22

  - name: Install dependencies
    run: npm install     #Executes a shell command

  - name: Run tests
    run: npm test

  - name: Build application
    run: npm run build
```