|  |  |
| --- | --- |
| **Topic** | DevOps |
| **Document Name** | DEVOPS-EX-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Difficulty Level** | | | |
| **Beginner** | **Junior** | **Senior** | **Expert** |
| □ | ■ | □ | □ |

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Ver | Comments |
| 01.07.2025 | Alican Çağdaş | 1.0 | Initial Draft |
| 19.07.2025 | Mennan Tekbir | 1.1 | Revisions |

# DevOps

## Exercise DEVOPS-EX-01:

Please answer the questions below

1. Please describe Configuration as Code (CaC) and Infrastructre as Code (IaC) with an example

 **Infrastructure as Code (IaC):** Managing and provisioning infrastructure (servers, networks, storage, etc.) using machine-readable configuration files rather than manual processes.  
**Example:** Using Terraform to define and create AWS EC2 instances and VPCs automatically.

 **Configuration as Code (CaC):** Managing the configuration and setup of software, services, and applications on infrastructure using code, ensuring consistent and automated setup.  
**Example:** Using Ansible playbooks to install and configure Nginx on servers.

1. What are the main differences between tools like Terraform, Ansible, and CloudFormation?

| **Feature** | **Terraform** | **Ansible** | **CloudFormation** |
| --- | --- | --- | --- |
| **Purpose** | Infrastructure provisioning (IaC) | Configuration management & automation | Infrastructure provisioning (IaC), AWS-specific |
| **Approach** | Declarative, tracks state file | Declarative & imperative, agentless | Declarative, AWS native |
| **Scope** | Multi-cloud (AWS, Azure, GCP, etc.) | Server config, app deployment | AWS resources only |
| **Functionality** | Creates and manages infrastructure | Installs packages, manages services | Defines AWS resources via JSON/YAML |
| **State Management** | Yes (state file) | No | Yes (managed by AWS) |

1. What is Docker Compose and what are its main use cases?

 **Docker Compose** is a tool to define and run multi-container Docker applications using a single YAML file (docker-compose.yml).

 **Use cases:**

* Running multi-service apps (e.g., frontend, backend, database) locally.
* Setting up dependent services quickly for development or testing.
* Automating environment setup in CI pipelines.

1. What is the default network type in Docker Compose, and how do services communicate with each other by default?

 **Default network type:** A user-defined bridge network automatically created per Compose project.

 **Service communication:** Services can communicate using their service names as DNS hostnames (e.g., http://backend:5000).

1. Given the following partial docker-compose.yml file, fill in the missing parts to define a simple web application with one backend (using the python:3.9 image) and one frontend (using the nginx:latest image) service. The backend should be accessible to the frontend on port 5000.

version: '3.8'  
 services:  
   backend:  
 image: python:3.9  
 command: python -m http.server 5000

ports:

- "5000:5000"  
  
   frontend:  
 image: nginx:latest  
 ports:

- "80:80"

depends\_on:

- backend

1. What is the difference between Continuous Integration (CI) and Continuous Deployment (CD)?

 **CI (Continuous Integration):** Developers frequently merge code changes into a central repository where automated builds and tests run to detect issues early.

 **CD (Continuous Deployment/Delivery):** After CI, the code changes are automatically (deployment) or semi-automatically (delivery) released to staging or production environments.

1. What is a pipeline? In a typical CI/CD pipeline, what steps would you include to ensure code quality and safe deployment?

* **Pipeline:** An automated sequence of steps (build, test, deploy) executed in software delivery.
* **Typical steps:** 
  1. **Checkout code**
  2. **Build/compile**
  3. **Static code analysis (linting)**
  4. **Unit tests**
  5. **Integration tests**
  6. **Packaging**
  7. **Deployment to test/staging/production**
  8. **Notifications and monitoring**

1. How would you configure a pipeline to deploy only when code is merged to the main branch, but run tests on every pull request?

 Set the pipeline trigger to run tests on **every pull request** event.

 Set deployment jobs to run **only on pushes to the main branch**.

 Example GitHub Actions snippet:

on:

pull\_request:

branches: [main]

push:

branches: [main]

jobs:

test:

runs-on: ubuntu-latest

steps:

# test steps here

deploy:

needs: test

if: github.ref == 'refs/heads/main' && github.event\_name == 'push'

steps:

# deploy steps here

1. You have deployed Prometheus and Grafana using Docker Compose to monitor your application. However, when you open Grafana, you cannot see any metrics from Prometheus. List possible reasons why Grafana cannot display metrics from Prometheus in this setup and suggest concrete steps or configuration changes to resolve this issue.

**Possible reasons:**

* + Grafana’s Prometheus data source URL is incorrect.
  + Prometheus container is not running or misconfigured.
  + Network connectivity issues between Grafana and Prometheus.
  + Prometheus data source is not properly added/configured in Grafana.

**Fixes:**

* + - Check docker-compose.yml exposes Prometheus port (usually 9090).
    - Configure Grafana data source URL as http://prometheus:9090.
    - Ensure both containers are on the same Docker network.
    - Check Prometheus logs (docker-compose logs prometheus) for errors.
    - Add and configure Prometheus as a data source in Grafana properly.

1. List 6 commonly used DevOps tools and briefly describe them.

* **Jenkins:** Open-source automation server for CI/CD pipelines.
* **Docker:** Containerization platform for packaging apps and dependencies.
* **Kubernetes:** Container orchestration platform for scaling and managing containers.
* **Terraform:** IaC tool to provision infrastructure across multiple cloud providers.
* **Ansible:** Automation and configuration management tool.
* **Prometheus:** Monitoring system and time series database.

11.Regarding to Branching strategies in Version Control Systems, which would you prefer and why?

* **Recommended:** Git Flow or GitHub Flow depending on project size.
* **Git Flow:** Good for complex projects with release management needs. Uses develop, feature, release, hotfix branches.
* **GitHub Flow:** Lightweight, ideal for continuous deployment with fewer branches.
* **Reason:** Strategy choice depends on team workflow, project complexity, and release cadence.

1. Write 10 widely used git commands and describe them

| **Command** | **Description** |
| --- | --- |
| git clone | Copy a remote repository locally. |
| git add | Stage changes for commit. |
| git commit | Save staged changes to local repo history. |
| git push | Upload commits to remote repository. |
| git pull | Fetch and merge changes from remote repo. |
| git status | Show working directory status. |
| git branch | List or create branches. |
| git checkout | Switch branches or restore files. |
| git merge | Combine changes from different branches. |
| git log | Show commit history. |

## Solution of DEVOPS-EX-01:

Please answer all question.

Create a LinkedIn post for the answer of question MODULO[calculator](https://www.omnicalculator.com/math/remainder)( your phone number, 12 ) + 1. For instance, if your phone number is 5339635384 please make a post for question 5 that describes your answer with additional info, image, video, links etc.

Best of Luck