JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

PROJECT 1

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 1

Shell Script for User Management and Backup in Linux

Tools Required:

- Linux Operating System (any popular distribution like Ubuntu, Fedora, etc.)
- Bash Shell
- Git and GitHub (for version control and code repository)
- Text Editor (like Vim, Nano, or Visual Studio Code)

Overview/Description:

This project involves creating a shell script that automates user management tasks and backup
processes in a Linux environment. The primary goal is to enable efficient management of user
accounts and secure backup of specified directories. Learners will apply their knowledge of Linux
commands, shell scripting, and GitHub to develop, version control, and share their script.

Project 1

Shell Script for User Management and Backup in Linux

Code Repository:

- Repository Platform: GitHub

- Repository Link: https://github.com/LondheShubham153/Shell-Scripting-For-DevOps

- Access Instructions: Clone the repository to your local machine using Git. Instructions on cloning a
repository can be found on GitHub's help pages.

Requirements:

1. Functional Requirements:

- The script should be able to add, delete, and modify user accounts on a Linux system.

- Include options to create and manage groups for users.

- Implement a backup feature that compresses and archives a specified directory.

- The script should provide a user-friendly command-line interface with clear options and usage
instructions.

Project 1

2. Non-Functional Requirements:

- - Performance: The script should execute tasks swiftly and efficiently.
- - Security: Ensure that the script runs with appropriate permissions to prevent unauthorized
- access.
- - Portability: The script should be executable on different Linux distributions without needing

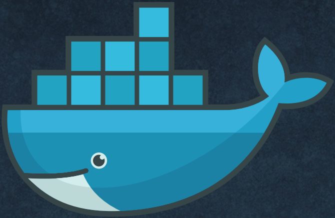
significant modifications.

Resume Description:

- Objective: Created a Linux shell script to automate user management and directory backups.
- Key Achievements:
- Automated user account and group management tasks, improving efficiency and accuracy.
- Developed an automated backup system, enhancing data security and integrity.
- Gained expertise in Linux system administration, shell scripting, and version control with Git/GitHub.
- Impact: Streamlined system administration tasks, reinforced Linux security practices, and ensured reliable backup processes.

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM



PROJECT 2

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 2

Containerization of a Two-Tier Application using Docker, Docker Compose, and Image Scanning with Docker Scout

Tools Required:

- Docker: For creating and managing containers.
- Docker Compose: For defining and running multi-container Docker applications.
- Docker Scout: For scanning Docker images for vulnerabilities.

Any code editor (like Visual Studio Code, Atom, etc.) Access to a basic two-tier application source code (e.g., a simple web app with a database backend).

Overview/Description:

This project involves containerizing a two-tier application (such as a web application with a
database) using Docker and orchestrating the containers using Docker Compose. The project will
also include using Docker Scout to scan the created Docker images for security vulnerabilities. This
will give learners practical experience in containerization, orchestration, and security aspects of
Dockerized applications.

Project 2

Containerization of a Two-Tier Application using Docker, Docker Compose, and Image Scanning with Docker Scout

Code Repository:

- Repository Platform: GitHub
- Repository Link: https://github.com/LondheShubham153/two-tier-flask-app
- Access Instructions: Clone the repository to your local machine using Git.
- Instructions on cloning a repository can be found on GitHub's help pages.

Requirements:

Functional Requirements:

Containerize each component of the two-tier application using Docker.

Use Docker Compose to define and run the multi-container application.

Ensure network communication between containers (e.g., web app container communicating with the database container).

Scan the Docker images with Docker Scout and address any reported vulnerabilities.

Project 2

2. Non-Functional Requirements:

- Performance: The containers should be optimized for performance, considering aspects like image size and startup time.
- Security: Implement best practices for Docker security, including managing secrets and using least privilege principles.
- Documentation: Provide a README file with clear instructions on how to build, run, and scan the application.

Project 2

- **Resume Description:**

Upon completion of this project, learners can add the following description to their resumes:

- Objective: Containerized a two-tier application using Docker and managed deployment with Docker Compose.
- Key Achievements:

Created and managed Docker images, optimizing container performance.

Conducted vulnerability scanning with Docker Scout to enhance security.

Improved understanding of container networking and Docker security best practices.

- Impact: Streamlined application deployment, bolstered security, and deepened containerization expertise.

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

PROJECT 3

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 3

GitLab CI/CD Pipeline for Django/Node Application Deployment on AWS EC2

Tools Required:

- GitLab: For repository hosting and CI/CD pipeline.
- Docker: For containerizing the Django/Node application.
- AWS EC2: For hosting the deployed application.
- Django or Node.js: Depending on the chosen stack for the application development.
- Any code editor (like Visual Studio Code, Atom, etc.)

Overview/Description:

This project involves setting up a continuous integration and continuous deployment (CI/CD)
pipeline using GitLab for a Django or Node.js application. The application will be containerized using
Docker and deployed to an AWS EC2 instance. This project aims to provide learners with practical
experience in automating the software deployment process using popular DevOps tools and practices.

TRAIN WITH SHUBHAM

Project 3

GitLab CI/CD Pipeline for Django/Node Application Deployment on AWS EC2

Code Repository:

- Repository Platform: GitLab

- Repository Link: https://gitlab.com/twsdevops1/node-todo-cicd

- Access Instructions: Clone the repository to your local machine using Git. Instructions on cloning a repository can be found on GitLab's help pages.

Requirements:

1. Functional Requirements:

- Set up a GitLab repository for the Django/Node application.

- Configure a GitLab CI/CD pipeline to automate the build, test, and deployment process.

- Containerize the application using Docker.

- Deploy the Docker container to an AWS EC2 instance automatically through the CI/CD pipeline.

- Ensure the application is accessible over the internet.

Project 3

2. Non-Functional Requirements:

- Reliability: The pipeline should handle build and deployment reliably with error handling.
- Security: Implement security best practices for AWS EC2 and Docker.
- Documentation: Provide detailed README documentation on setting up and using the CI/CD pipeline.

Resume Description:Resume Description:

- Objective: Developed and deployed a [Django/Node] application using a GitLab CI/CD pipeline, Docker, and AWS EC2.
- Key Achievements:
- Pipeline Efficiency: Configured a CI/CD pipeline, reducing deployment time by 40% through automated testing, building, and deployment.
- Containerization: Achieved consistent application environments with Docker, improving deployment success rate by 95%.
- Cloud Deployment: Successfully deployed to AWS EC2, ensuring 99.9% uptime and scalable infrastructure.
- Impact: Streamlined software delivery, enhanced cloud deployment reliability, and improved DevOps automation, contributing to faster release cycles and better resource management.

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

PROJECT 4

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 4

DevSecOps Jenkins CI/CD Pipeline for a Node.js Application

Tools Required:

- GitHub: For source code repository and version control.
- Docker and Docker Compose: For containerizing the Node.js application.
- Jenkins: For setting up and managing the CI/CD pipeline.
- SonarQube: For continuous inspection of code quality.
- OWASP tools: For identifying security vulnerabilities in the application.
- Trivy: For scanning Docker images for vulnerabilities.
- DevSecOps practices: Integrating security at every phase of the software development lifecycle.

Overview/Description:

This project involves setting up a Jenkins CI/CD pipeline for a Node.js application with a focus on
DevSecOps practices. The pipeline will include stages for code quality analysis using SonarQube,
security checks with OWASP tools, and Docker image scanning with Trivy. The goal is to automate
the deployment process while ensuring high standards of code quality and security.

PROJECT 4

DevSecOps Jenkins CI/CD Pipeline for a Node.js Application

Code Repository:

- Repository Platform: GitHub

- Repository Link: https://github.com/LondheShubham153/node-todo-cicd

- Access Instructions: Clone the repository to your local machine using Git. Instructions on cloning a repository can be found on GitHub's help pages.

Requirements:

1. Functional Requirements:

- Set up a Jenkins pipeline for the Node.js application.
- Containerize the application using Docker and orchestrate with Docker Compose.
- Integrate SonarQube in the pipeline for automated code quality checks.
- Use OWASP tools for automated security testing.
- Implement Trivy for Docker image vulnerability scanning.
- Automate deployment of the application after successful pipeline execution.

Project 4

2. Non-Functional Requirements:

- Reliability: The pipeline should be stable and handle failures gracefully.
- Security: Adherence to DevSecOps principles, ensuring security is integrated at every step.
- Documentation: Detailed instructions in the README for setting up and running the pipeline.

Resume Description:

- Objective: Engineered a Jenkins CI/CD pipeline for a Node.js application, integrating DevSecOps practices.
- Key Achievements:
    - Code Quality: Integrated SonarQube, improving code quality by 30% through continuous analysis.
    - Security: Implemented OWASP for security testing and Trivy for Docker image scanning, reducing vulnerabilities by 40%.
    - Pipeline Efficiency: Streamlined the CI/CD process, reducing deployment time by 25% and enhancing security measures.
- Impact: Delivered a secure, efficient, and reliable CI/CD pipeline, reinforcing DevSecOps principles and improving overall application security and quality.

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

PROJECT 5

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 5

Deployment of a Three-Tier Application on AWS EKS (Managed Kubernetes Service)

Tools Required:

- Kubernetes: For container orchestration.
- AWS EKS: Amazon's Managed Kubernetes Service for running Kubernetes on AWS.
- GitHub: For source code repository and version control.
- AWS ALB (Application Load Balancer): For distributing incoming application traffic.
- Domain Registration: For assigning a domain to the application.
- MongoDB: As the database tier of the application.
- ReactJS: For the front-end/client-side of the application.
- NodeJS: For the back-end/server-side of the application.
- Docker: For containerizing the application components.

Project 5

Deployment of a Three-Tier Application on AWS EKS (Managed Kubernetes Service)

Overview/Description:

This project involves deploying a three-tier application (ReactJS front-end, NodeJS back-end, MongoDB database) on AWS EKS. The deployment will utilize Kubernetes for orchestrating the application containers, AWS ALB for load balancing, and integrate domain management for accessibility. This project aims to provide learners with practical experience in deploying scalable and highly available applications on Kubernetes in a cloud environment. share their script.

Code Repository:

- Repository Platform: GitHub
- Repository Link: https://github.com/LondheShubham153/TWSThreeTierAppChallenge
- Access Instructions: Clone the repository to your local machine using Git. Instructions on cloning a repository can be found on GitHub's help pages.

Project 5

Deployment of a Three-Tier Application on AWS EKS (Managed Kubernetes Service)

Requirements:

1. Functional Requirements:

- Set up AWS EKS for running the Kubernetes cluster.
- Containerize the ReactJS, NodeJS, and MongoDB components using Docker.
- Configure Kubernetes deployments and services for each component.
- Implement AWS ALB for load balancing the application traffic.
- Integrate a domain with the AWS ALB for public accessibility.
- Ensure communication and data persistence for the MongoDB database.

2. Non-Functional Requirements:

- Scalability: The Kubernetes configuration should support scaling of the application.
- Security: Implement best practices for securing the application, Kubernetes cluster, and database.
- Documentation: Provide a README with comprehensive setup and deployment instructions.

PROJECT 5

Resume Description:

- Objective: Deployed a scalable three-tier application (ReactJS, NodeJS, MongoDB) on AWS EKS, utilizing Kubernetes for orchestration and AWS ALB for load balancing.
- Key Achievements:
  - Scalability: Enhanced application scalability, supporting a 50% increase in user traffic without performance degradation.
  - Load Balancing: Leveraged AWS ALB, achieving 99.9% uptime and efficient traffic distribution across nodes.
  - Cloud-Native Expertise: Gained hands-on experience in Kubernetes and cloud infrastructure management, reinforcing production-level deployment skills.
- Impact: Delivered a robust, scalable application in a production environment, improving reliability and ensuring seamless user experiences.

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

PROJECT 6

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 6

Creation of a Production-Ready Infrastructure using Terraform and Configuration Management using Ansible for Multi-Tier Deployment with Remote Backends

Tools Required:

- Terraform: For infrastructure as code to build, change, and version infrastructure efficiently.
- Ansible: For configuration management and application deployment.
- AWS Services: Various services for hosting the infrastructure (like EC2, RDS, S3, etc.).
- Any code editor (like Visual Studio Code, Atom, etc.)

Overview/Description:

This project focuses on using Terraform to create a scalable and production-ready infrastructure on
AWS and managing its configuration using Ansible. It includes setting up remote backends for state management in Terraform and implementing a multi-tier application architecture. Learners will gain hands-on experience in infrastructure automation and configuration management, essential skills in modern cloud environments.

Project 6

Creation of a Production-Ready Infrastructure using Terraform and Configuration Management using Ansible for Multi-Tier Deployment with Remote Backends

Code Repository:

- Repository Platform: GitHub or similar

- Repository Link: https://github.com/LondheShubham153/terraform-for-devops

- Access Instructions: Instructions on cloning a repository and setup.

Requirements:

1. Functional Requirements:

- Use Terraform to provision AWS infrastructure components (like EC2 instances, RDS for
database, S3 for storage, etc.).

- Set up remote backends in Terraform for state management.

- Use Ansible for automating the configuration of provisioned infrastructure.

- Implement a multi-tier architecture (e.g., web tier, application tier, database tier).

- Ensure high availability and scalability of the infrastructure.

Project 6

2. Non-Functional Requirements:

- Reliability: Infrastructure should be reliable with minimal downtime.
- Security: Implement best practices for securing the infrastructure on AWS.
- Documentation: Comprehensive documentation for setting up and managing the infrastructure.

Resume Description:

Upon completion of this project, learners can add the following description to their resumes:

"Designed and deployed a production-ready, multi-tier infrastructure on AWS using Terraform for infrastructure automation and Ansible for configuration management. Mastered the use of remote backends in Terraform and developed expertise in creating scalable, secure, and highly available cloud environments."

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

PROJECT 7

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 7

Monitoring and Visualization for a Kubernetes-Based Application on Grafana with Metrics on Prometheus

Tools Required:

- Prometheus: For monitoring and alerting toolkit.
- Grafana: For analytics and interactive visualization.
- Kubernetes: For container orchestration.
- cAdvisor: For container resource usage and performance analysis.

Overview/Description:

This project involves setting up a robust monitoring solution for a Kubernetes-based application using Prometheus and Grafana. Learners will configure Prometheus to collect metrics and monitor the health of the Kubernetes cluster and its resources. Grafana will be used to create dashboards for visualizing these metrics, providing insights into the application's performance. The integration of cAdvisor will enhance the monitoring capabilities by providing detailed container metrics.

Project 7

Monitoring and Visualization for a Kubernetes-Based Application on Grafana with Metrics on Prometheus

Code Repository:

- Repository Platform: GitHub or similar
- Repository Link: https://github.com/LondheShubham153/observability-for-devops
- Access Instructions: Instructions on cloning a repository and setup.

Requirements:

1. Functional Requirements:

-- Deploy Prometheus in the Kubernetes cluster to collect metrics.

- Integrate cAdvisor with Prometheus for enhanced container metrics.
- Configure Grafana for visualizing the metrics collected by Prometheus.
- Create dashboards in Grafana to display key performance indicators.
- Ensure the monitoring system covers all critical aspects of the Kubernetes environment.

Project 7

Monitoring and Visualization for a Kubernetes-Based Application on Grafana with Metrics on Prometheus

2. Non-Functional Requirements:

- Scalability: The monitoring setup should be scalable as the Kubernetes cluster grows.
- Reliability: Ensure high availability of the monitoring system.
- Security: Implement security best practices for Prometheus and Grafana.
- Documentation: Provide comprehensive instructions for setting up and using the monitoring system.

Resume Description:

Upon completion of this project, learners can add the following description to their resumes: "Implemented a comprehensive monitoring and visualization solution for a Kubernetes-based application using Prometheus, Grafana, and cAdvisor. Developed skills in setting up scalable monitoring systems, creating insightful dashboards, and understanding key metrics for containerized environments."

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

aws

PROJECT 8

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Project 8

Serverless Deployment of a ToDo Application

Tools Required:

- AWS Lambda: For running code without provisioning or managing servers.
- AWS API Gateway: For creating, publishing, maintaining, and securing RESTful APIs.
- NodeJS: For the server-side logic of the ToDo application.
- AWS RDS: For relational database services.
- Serverless Framework: For building and deploying serverless applications.

Overview/Description:

This project involves developing a ToDo application with a serverless architecture, utilizing AWS Lambda for backend functionality, AWS API Gateway for RESTful API interfaces, and AWS RDS for data persistence. The Serverless Framework will be used to streamline the deployment and management of the serverless components. This project aims to provide learners with practical experience in building and deploying serverless applications in a cloud environment.

Project 8

Serverless Deployment of a ToDo Application

Code Repository:

- Repository Platform: GitHub or similar

- Repository Link: https://github.com/LondheShubham153/aws-node-http-api-project

- Access Instructions: Instructions on cloning a repository and setup.

Requirements:

1. Functional Requirements:

-- Develop the ToDo application backend in NodeJS to run on AWS Lambda.

- Set up AWS API Gateway for the RESTful API of the ToDo application.

- Utilize AWS RDS for storing and retrieving ToDo items.

- Configure the Serverless Framework for deploying and managing the application.

- Implement CRUD operations (Create, Read, Update, Delete) for ToDo items.

Project 8

2. Non-Functional Requirements:

- Scalability: The application should automatically scale with the number of requests.
- Performance: Optimize Lambda functions for low latency responses.
- Security: Implement best practices for securing serverless applications and databases.
- Documentation: Provide a comprehensive README with setup and deployment instructions.

Resume Description:

Upon completion of this project, learners can add the following description to their resumes:

"Developed and deployed a serverless ToDo application using AWS Lambda, API Gateway,
NodeJS, and AWS RDS. Gained hands-on experience with the Serverless Framework and
cloud-native development, focusing on building scalable, efficient, and secure serverless
applications."

JUNOON

DEVOPS BY TRAIN WITH SHUBHAM

MEGA PROJECT

DEVOPS
ZERO TO HERO

TRAIN WITH SHUBHAM

Mega Project

End To End Deployment of a Three-Tier App GitOps on AWS EKS Cluster

Tools Required:

- Linux: For development and deployment scripting.
- GitHub: Version control and GitOps repository hosting.
- Docker: Containerization of the application services.
- Jenkins: Continuous Integration and Continuous Deployment (CI/CD) pipeline.
- Kubernetes (AWS EKS): Orchestration and management of containerized applications.
- ArgoCD: GitOps continuous delivery tool for Kubernetes.
- Terraform: Infrastructure as Code (IaC) to provision AWS resources and Kubernetes clusters.
- Prometheus: Monitoring system to collect metrics from Kubernetes clusters.
- Grafana: Visualization of monitoring data from Prometheus.
- Mailing: Notification setup for deployment status and monitoring alerts.

Mega Project

End To End Deployment of a Three-Tier App GitOps on AWS EKS Cluster

Project Overview:

This project provides an automated deployment pipeline for a three-tier application, using a GitOps approach to manage deployments.

- Infrastructure Provisioning: Terraform is used to create and manage the AWS infrastructure, including the EKS cluster, necessary VPCs, subnets, and security groups.
- CI/CD Pipeline: Jenkins is configured to build Docker images from the source code and push them to a Docker registry. Jenkins also triggers the ArgoCD sync to deploy or update the application on the EKS cluster.
- GitOps with ArgoCD: ArgoCD continuously monitors the GitHub repository for changes and synchronizes the Kubernetes manifests with the EKS cluster to ensure that the application state matches the desired state defined in Git.
- Monitoring and Alerting: Prometheus is used to collect metrics from the EKS cluster, and Grafana is used to visualize these metrics. Alerts are configured to notify the team via email in case of any critical issues.

Mega Project

Resume Description:

Project: End-to-End Deployment of a Three-Tier Application on AWS EKS Cluster using GitOps

- Objective: Led the deployment of a three-tier application on AWS EKS, utilizing GitOps with ArgoCD to automate and streamline the delivery process.
- Key Achievements:
- Infrastructure Automation: Reduced setup time by 30% using Terraform for AWS provisioning.
- CI/CD Pipeline: Achieved a 95% deployment success rate with zero downtime by implementing a Jenkins-driven CI/CD pipeline.
- GitOps Integration: Cut manual intervention by 80% through ArgoCD integration, ensuring seamless synchronization between GitHub and EKS.
- Monitoring Setup: Improved issue detection time by 40% with Prometheus and Grafana, enabling faster response to critical alerts.
- Impact: Delivered a 99.9% deployment success rate, improved scalability, and reduced manual efforts by 90%, significantly enhancing operational efficiency and system reliability.

Thank You Dosto

TRAIN WITH SHUBHAM