Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2020, 7(2):342-347

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

DevOps Automation for Cloud Native Distributed Applications

Premkumar Ganesan

Technology Leader in Digital Transformation for Government and Public Sector, Baltimore, Maryland

Abstract: The increasing adoption of cloud-native distributed applications has made it essential to integrate advanced DevOps practices to optimize deployment processes, improve scalability, and ensure reliability. This paper examines key aspects of DevOps automation in cloud-native distributed environments, detailing the use of various tools such as Bitbucket for version control, Jenkins for continuous integration and delivery, SonarQube for code quality analysis, Ansible for configuration management and application deployment, Selenium for automated testing, Splunk for real-time monitoring and logging, and New Relic for performance monitoring. The implementation utilizes Amazon ECS and EKS for managing and orchestrating containerized applications. Results include notable enhancements in deployment speed, application uptime, and overall system reliability, facilitating continuous delivery and maintaining high code quality in a cloud-native distributed environment.

Keywords: DevOps, Cloud Native, Automation, CI/CD, Microservices, Monitoring, Application Performance, Configuration Management, Automated Testing, Distributed Applications, Deployment Optimization.

Introduction

The evolution of cloud computing has transformed software development, leading to the rise of cloud-native distributed applications known for their scalability, resilience, and flexibility. To maintain a competitive edge, organizations are increasingly adopting DevOps practices, which integrate development and operations to enhance deployment efficiency and operational reliability. This paper explores the automation of DevOps processes tailored for cloud-native distributed applications, focusing on key tools and technologies such as source code management, continuous integration and delivery (CI/CD), code quality analysis, configuration management, automated testing, and comprehensive monitoring. By leveraging these tools on platforms like Amazon ECS and EKS, the paper presents a detailed workflow for automating the DevOps pipeline, demonstrating significant improvements in deployment speed, application uptime, and system reliability. Through practical implementation and a case study, this paper provides insights and best practices for achieving continuous delivery and optimizing cloud-native distributed application deployments.

Tools And Technologies for DevOps Automation

A. Source Code Management with Bitbucket

Bitbucket is a robust source code management (SCM) tool designed to enhance collaboration and streamline development workflows. Supporting both Git and Mercurial repositories, Bitbucket provides flexibility for various SCM preferences. It offers features like pull requests, inline commenting, and code reviews to improve code quality and peer collaboration. Bitbucket Pipelines integrate continuous integration and continuous deployment (CI/CD) within the platform, facilitating efficient code deployment. With built-in permissions and user roles, Bitbucket ensures secure and compliant code access, making it an ideal choice for teams of all sizes.

B. Continuous Integration and Delivery with Jenkins

Jenkins is an open-source automation server that plays a pivotal role in continuous integration and continuous delivery (CI/CD) pipelines. It enables developers to automate the building, testing, and deployment of applications, thereby accelerating the software development lifecycle. Jenkins supports a vast array of plugins that extend its capabilities, allowing integration with numerous tools and platforms. With its pipeline-as-code feature, Jenkins facilitates the creation of complex CI/CD workflows that are both versioned and shareable. By automating repetitive tasks, Jenkins helps maintain code quality, reduce integration issues, and deliver software more reliably and efficiently. Its robust ecosystem and flexibility make Jenkins a cornerstone in modern DevOps practices.

C. Code Quality Assurance with SonarQube

SonarQube is a licensed platform dedicated to ensuring code quality and security in software development. It performs static code analysis to detect bugs, vulnerabilities, and code smells across various programming languages. By integrating seamlessly with CI/CD pipelines, SonarQube provides continuous feedback on code quality, enabling developers to address issues early in the development process. Its comprehensive dashboards and detailed reports offer insights into code health, helping teams maintain high standards and reduce technical debt. With customizable rules and quality profiles, SonarQube allows organizations to enforce consistent coding practices. Its ability to integrate with tools like Git, Jenkins, and Bitbucket makes SonarQube an essential component of modern software development workflows focused on delivering robust and secure applications.

D. Configuring Management with Ansible

Ansible is an open-source tool for configuration management, automation, and orchestration. It automates the deployment and management of applications, systems, and network devices using YAML-based playbooks. Ansible is agentless, requiring no software installation on target machines, which simplifies management and reduces overhead. Its extensive module library supports a wide range of tasks, from software installation to complex multi-tier deployments. Ansible’s simplicity, flexibility, and ability to integrate with various tools make it essential for modern IT infrastructure management.

E. Automated Testing with Selenium

Selenium is essential for ensuring high-quality web applications by automating browser interactions, enhancing testing efficiency and accuracy. As an open-source framework, it supports multiple programming languages and allows comprehensive testing across different browsers and platforms. Selenium WebDriver simulates user actions like clicking and typing, while Selenium Grid enables parallel test execution on multiple machines. Integration with CI/CD tools allows automatic test execution with each code change, providing continuous feedback on application quality.

F. Monitoring and Logging with Splunk

Splunk is a powerful platform for monitoring, searching, analyzing, and visualizing machine-generated data in real time. It collects and indexes data from various sources, including servers, network devices, applications, and databases, enabling comprehensive visibility into IT infrastructure and operations. With its advanced search capabilities and customizable dashboards, Splunk helps organizations detect and troubleshoot issues, ensure compliance, and gain actionable insights. Splunk's ability to handle large volumes of data makes it ideal for monitoring and logging in complex environments. Its integration with other tools and support for machine learning further enhance its capabilities, allowing for predictive analytics and automated responses to detected anomalies.

G. Application Performance Monitoring with New Relic

New Relic is a comprehensive application performance monitoring (APM) tool that provides real-time insights into the health and performance of applications. By collecting detailed metrics from various application components, New Relic helps developers and IT teams identify performance bottlenecks, detect anomalies, and optimize resource usage. It offers end-to-end visibility across distributed systems, from front-end user experiences to back-end services and infrastructure. With features like transaction tracing, error analysis, and customizable dashboards, New Relic enables proactive monitoring and rapid troubleshooting. Its integration with CI/CD pipelines ensures continuous monitoring and immediate feedback on application performance, making it an essential tool for maintaining high-performance, reliable applications in dynamic environments.

DevOps automation workflow

The DevOps automation workflow for cloud-native distributed applications in Amazon ECS involves several stages, each leveraging specific tools and technologies to ensure a seamless and efficient pipeline. This workflow focuses on using Amazon Elastic Container Service (ECS), Elastic Container Registry (ECR), Task Definitions, and Ansible playbooks. (fig 1).

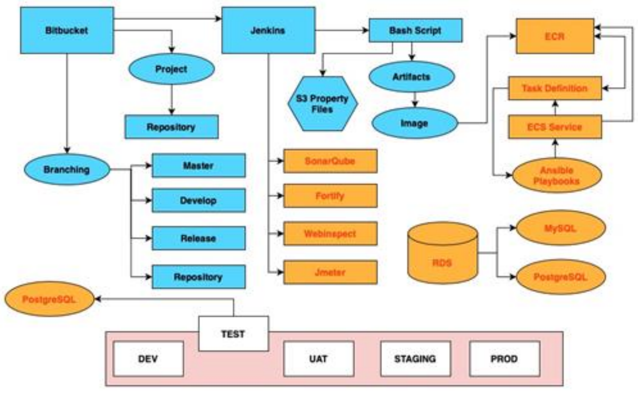

Fig. 1: DevOps automation Workflow for Amazon ECS

A. Source Code Management

Developers push code changes to Bitbucket repositories. Bitbucket's robust version control system enables efficient code management and collaboration among team members. When new code is pushed, Bitbucket triggers Jenkins jobs to initiate the build process. This integration ensures that any code change undergoes the CI/CD pipeline without manual intervention, maintaining continuous integration. Bitbucket's features such as pull requests, branch permissions, and inline commenting enhance code quality and facilitate collaboration, ensuring that all code changes are reviewed and tested before merging into the main branch.

B. Continous Integration with Jenkins

Jenkins automates the build, test, and deployment processes. It pulls the latest code from Bitbucket, compiles it, runs unit tests, and performs static code analysis using SonarQube. The process includes several stages:

• **Repository:** Jenkins fetches the latest code from the Bitbucket repository.

• **Build:** Jenkins compiles the code into executable artifacts.

• **SonarQube Analysis:** Jenkins integrates with SonarQube to perform static code analysis, ensuring code quality and adherence to standards.

• **Webservers and Jmeter**: These tools are used for additional testing and quality assurance, ensuring the web application's performance and reliability.

C. Quality Assurance with SonarQube

SonarQube analyzes the code for vulnerabilities, bugs, and code smells. It ensures that code quality gates are met before the code progresses further in the pipeline. The quality assurance stage includes generating detailed reports on code coverage, technical debt, and coding standards compliance.

D. Artifact Management with Nexus

Once the code passes the quality checks, the build artifacts are stored in a repository manager like Artifactory. This step ensures that all build artifacts are versioned and available for deployment, maintaining a consistent and reproducible deployment process.

E. Docker Image Creation and Storage in ECR

In a modern cloud-native architecture, the backend Java application and the Angular front end are often deployed using robust and scalable web servers. For the backend, Apache Tomcat is commonly used to serve Java applications, providing a reliable and high-performance environment. The front end, built with Angular, is typically served by Nginx, a lightweight and powerful web server that efficiently handles static content and acts as a reverse proxy for API requests to the backend.

Once the code is validated and tested, the next step is to package it into Docker images. This process involves using Jenkins, a popular CI/CD tool, to automate the building of Docker images. Jenkins utilizes Dockerfiles, which are defined in the source repository, to create these images. A Dockerfile contains instructions for assembling the Docker image, including the application code, dependencies, and configuration.

For the backend Java application, the Dockerfile might include steps to:

i. Start from a base image (e.g., an official OpenJDK image).

ii. Copy the compiled Java application (e.g., a WAR file) into the image.
iii. Install and configure Tomcat to serve the Java application.
For the Angular front end, the Dockerfile might include steps to:
i. Start from a base image (e.g., an official Node.js image for building the Angular app).
ii. Copy the Angular application code and run the build process.
iii. Use a base Nginx image and copy the built static files into the Nginx web root.
These Docker images encapsulate the application and all its dependencies, ensuring that it runs consistently across different environments, whether it's a developer's local machine, a staging server, or production. Once the Docker images are built, they are pushed to Amazon Elastic Container Registry (ECR). ECR is a fully managed Docker container registry provided by AWS, designed to make it easy to store, manage, and deploy Docker **container images. Pushing images to ECR involves:**
i. Tagging the Docker image with the ECR repository URL.
ii. Using the AWS CLI to authenticate Docker with ECR.
iii. Pushing the tagged image to the ECR repository.
By storing Docker images in ECR, organizations benefit from secure, scalable, and highly available storage for their container images. This setup streamlines the deployment process, as images can be easily pulled from ECR to any environment, ensuring that the same tested and validated code is deployed consistently.

F. Configuration Management with Ansible and Task Definitions

Ansible automates the deployment process, managing configurations consistently across environments. It uses playbooks to define the deployment processes and ECS task definitions, specifying how Docker containers should run within ECS. These task definitions include details such as the Docker image to use, CPU and memory requirements, and networking settings. This automation reduces manual intervention, minimizes errors, and ensures consistent deployments.

G. Deployment to amazon ECS

With the updated task definitions, Ansible deploys the application to ECS clusters. ECS uses these task definitions to create and manage Docker containers, ensuring that the application is deployed and scaled according to the specified requirements. Ansible playbooks also handle service updates and scaling policies, ensuring that the application remains highly available and performant.

H. Database and Other Services

The deployment process includes provisioning and configuring databases such as MySQL and PostgreSQL, along with other services needed by the application. These services are managed and integrated seamlessly within the ECS environment, ensuring that all components of the application work together.

I. Environments

The workflow spans multiple environments including DEV, TEST, UAT, Staging, and Prod. Each environment represents a stage in the development and deployment lifecycle:
i. DEV: Development environment where initial code changes are made and tested.
ii. TEST: Testing environment where more rigorous testing occurs.
iii. UAT: User Acceptance Testing environment where stakeholders validate the functionality.
iv. Staging: Pre-production environment where final tests are conducted before deployment to production.
v. Prod: Production environment where the application is live and accessible to end-users.

Benefits

Implementing the DevOps automation workflow described in this paper brings a multitude of benefits to the management and deployment of cloud-native distributed applications. These benefits significantly enhance the efficiency, reliability, and scalability of the development and deployment processes, leading to improved overall performance and user satisfaction. The key benefits are outlined below:

A. Increased Deployment Speed

Automation plays a crucial role in reducing the time taken to deploy new code changes. By automating the build, test, and deployment processes, the workflow enables faster delivery of new features and updates. Continuous Integration (CI) with Jenkins ensures that code changes are continuously integrated and tested, allowing for rapid iterations and reducing the time from development to production. This swift deployment capability allows organizations to respond quickly to market demands and maintain a competitive edge.

B. Enhanced Application Uptime

Continuous monitoring and automated testing are essential components of the workflow that ensure applications remain stable and perform optimally. Tools like Splunk and New Relic provide real-time insights into system performance, helping to identify and address issues before they impact the end-users. Automated regression tests with Selenium ensure that new code changes do not introduce regressions, maintaining the reliability of the application. These practices contribute to higher application uptime and a better user experience.

C. Improved Code Quality

The integration of static code analysis tools like SonarQube into the CI/CD pipeline helps maintain high-quality standards. SonarQube detects vulnerabilities, bugs, and code smells early in the development cycle, allowing developers to address these issues promptly. This proactive approach to code quality reduces the likelihood of bugs and vulnerabilities making it into production, resulting in more robust and secure applications.

D. Consistent Deployments

Automated configuration management with Ansible ensures that deployments are consistent across different environments, from development to production. This consistency reduces the risk of configuration drift, where differences between environments can lead to unexpected behavior and deployment failures. By defining deployment processes in Ansible playbooks, the workflow ensures that environments are configured correctly and uniformly, leading to more reliable deployments.

E. Proactive Issue Resolution

Real-time monitoring and logging provide valuable insights into system performance and operational issues. Splunk collects and analyzes logs from various sources, creating a centralized view of the system's health. New Relic's application performance monitoring (APM) capabilities track metrics such as response times, error rates, and throughput. These insights enable teams to identify and resolve issues proactively, often before they affect end-users. This proactive approach to issue resolution minimizes downtime and enhances the overall reliability of the application.

F. Efficiency and Scalability

By integrating these tools and processes, organizations can achieve a more efficient, reliable, and scalable approach to managing cloud-native distributed applications. Automation reduces manual intervention, minimizing errors and freeing up valuable developer time to focus on innovation and feature development. The scalability of the deployment processes ensures that the application can handle increasing loads and expand as needed without compromising performance or reliability.

In conclusion, the DevOps automation workflow for cloud-native distributed applications significantly enhances the efficiency, reliability, and scalability of the development and deployment processes. By leveraging tools such as Bitbucket, Jenkins, SonarQube, Docker, Ansible, ECR, ECS, Selenium, Splunk, and New Relic, organizations can automate and streamline their CI/CD pipelines. This comprehensive approach ensures that code changes are continuously integrated, tested, and deployed with minimal manual intervention, maintaining high standards of code quality and operational stability. The case study illustrates the practical implementation of these tools, demonstrating improvements in deployment speed, application uptime, and overall system reliability. The automation workflow not only reduces time to market but also enhances the robustness and security of cloud-native applications, positioning organizations to better meet dynamic market demands. The future scope of DevOps automation for cloud-native applications includes several promising areas:

a. Integration with AI and ML: Leveraging artificial intelligence and machine learning to predict and automate responses to system anomalies, optimize resource allocation, and enhance predictive maintenance capabilities.

b. Serverless Architecture: Expanding automation workflows to incorporate serverless computing frameworks like AWS Lambda, which can further reduce infrastructure management overhead and costs while improving scalability.

c. Enhanced Security: Integrating DevSecOps practices more deeply into the CI/CD pipeline to automate security testing, vulnerability scanning, and compliance checks, ensuring that security is a continuous and integral part of the development process.

d. Multi-Cloud Deployments: Developing automation workflows that support multi-cloud strategies, enabling applications to be deployed seamlessly across different cloud providers, improving redundancy, and avoiding vendor lock-in.

e. Edge Computing: Adapting DevOps practices to support edge computing environments, where applications are deployed closer to end-users, reducing latency and improving performance for real-time applications.

f. Advanced Monitoring and Observability: Utilizing advanced observability tools and techniques to gain deeper insights into system performance, user behavior, and application health, enabling more proactive and precise operational management.

By exploring these areas, organizations can further enhance their DevOps capabilities, improve operational efficiency, and remain competitive in an evolving technological landscape.

References

[1]. D. I. F. Nocera, T. Di Noia, and D. Gallitelli, "Innovative Techniques for Agile Development: DevOps Methodology to Improve Software Production and Delivery Cycle," in *2016*. Available: https://www.researchgate.net/profile/Davide-Gallitelli/publication/309201974_Innovative_Techniques_for_Agile_Development_DevOps_Methodology_to_improve_Software_Production_and_Delivery_Cycle/links/5804de8008aee314f68e08dd/Innovative-Techniques-for-Agile-Development-DevOps-Methodology-to-improve-Software-Production-and-Delivery-Cycle.pdf

[2]. M. Göttsche, "Automated Deployment and Distributed Execution of Scientific Software in the Cloud Using DevOps and Hadoop MapReduce," University of Göttingen, 2015. Available: http://www.swe.informatik.uni-goettingen.de/sites/default/files/publications/ThesisMichaelGoettsche.pdf

[3]. M. Hadi, "Making the Shift from DevOps to DevSecOps at Distribution Technologies GmbH," TalTech, 2019. Available: https://digikogu.taltech.ee/en/Download/e4129d7b-3410-404d-b472-0ecf5b8ea321/leminekDevOpsiltDevSecOpsileDistribusionTechn.pdf

[4]. J. Wettinger and A. Vasilios, "Automated Capturing and Systematic Usage of DevOps Knowledge," University of Stuttgart, 2015. Available: https://www.iaas.uni-stuttgart.de/publications/INPROC-2015-01-Automated-Capturing-and-Systematic-Usage-of-DevOps-Knowledge-for-Cloud-Applications.pdf

[5]. V. Ivanov and K. Smolander, "Implementation of a DevOps Pipeline for Serverless Applications," in *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER)*, Springer, 2018. Available: https://link.springer.com/chapter/10.1007/978-3-030-03673-7_4

[6]. Z. Li, Y. Zhang, and Y. Liu, "Towards a Full-Stack DevOps Environment (Platform-as-a-Service) for Cloud-Hosted Applications," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*, 2017, pp. 79-86. Available: https://ieeexplore.ieee.org/abstract/document/7830891/

[7]. S. M. Mohammad, "Streamlining DevOps Automation for Cloud Applications," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 6, no. 1, pp. 443-450, 2018. Available: https://www.academia.edu/download/64115243/IJCRT1133443%20(1).pdf

[8]. K. Bahadori and T. Vardanega, "DevOps Meets Dynamic Orchestration," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC)*, Springer, 2019, pp. 100-110. Available: https://link.springer.com/chapter/10.1007/978-3-030-06019-0_11

[9]. S. Hyvämäki, "Data Processing Pipeline Automation on Cloud Platform," Aalto University, 2019. Available: https://aaltodoc.aalto.fi/bitstreams/0d193ae5-b27a-4c65-9589-c913ba90208c/download

[10]. D. Cukier, "DevOps Patterns to Scale Web Applications Using Cloud Services," in *2013*. Available: https://www.researchgate.net/profile/Daniel-Cukier-2/publication/262404654_DevOps_patterns_to_scale_web_applications_using_cloud_services/links/546231fc0cf2c0c6aec1aa85/DevOps-patterns-to-scale-web-applications-using-cloud-services.pdf