Pet Clinic

Automated Deployment Pipeline

Deirdre Lee

devops T00126583

Contents

[Stage 1: Code and Tools 2](#_Toc160541226)

[Stage 1.1: Summary of Application 2](#_Toc160541227)

[Stage 1.2: Tool Chain 2](#_Toc160541228)

[Stage 1.3: Flow Diagram 2](#_Toc160541229)

[Stage 2: Continuous Integration 2](#_Toc160541230)

[Stage 2.1: Install & Configure Continuous Integration 2](#_Toc160541231)

[Stage 2.2: Automate Build Process 2](#_Toc160541232)

[Stage 2.3: Configure Git Authentication 2](#_Toc160541233)

[Stage 2.4: Unit Test Execution 2](#_Toc160541234)

[Stage 2.5: Code Quality Tools 2](#_Toc160541235)

[Stage 3: Building the Code & Configuring the Pipeline 2](#_Toc160541236)

[Stage 3.1: Task Pipelines 2](#_Toc160541237)

[Stage 3.2: Deploy Application 3](#_Toc160541238)

[Stage 3.3: Build Pipeline for CI Lifecycle 3](#_Toc160541239)

[Stage 4: End-to-End Automation Of The Application Delivery Lifecycle 3](#_Toc160541240)

[Stage 5: Cloud Provisioning and Configuration Management 3](#_Toc160541241)

[Stage 6: Deploying Application (AWS, Azure, and Docker) 3](#_Toc160541242)

[Stage 7: Monitoring Infrastructure and Applications 3](#_Toc160541243)

[Stage 8: Orchestrating Application Deployment 3](#_Toc160541244)

# Stage 1: Code and Tools

## Stage 1.1: Summary of Application

* Describe what the application does.
* Specify where it was sourced.
* List the technologies used (e.g., Java, Spring).

## Stage 1.2: Tool Chain

* Add SVN, Git, Apache Maven, Jenkins, AWS, Microsoft Azure, and other DevOps technologies.
* Continuously update the list as the semester progresses.

## Stage 1.3: Flow Diagram

* Highlight the technologies to be implemented at each stage.

# Stage 2: Continuous Integration

## Stage 2.1: Install & Configure Continuous Integration

* Set up continuous integration using a sample application.
* Store the application in a repository like GitHub.

## Stage 2.2: Automate Build Process

* Explore automation of the build process from a repository (Git).
* Utilise tools such as Maven, Gradle, MSBuild, etc.

## Stage 2.3: Configure Git Authentication

* Ensure that new jobs in the CI server authenticate with Git.
* Configure Git within the CI server.

## Stage 2.4: Unit Test Execution

* Perform unit tests execution in the sample application.
* Step 5: Configure Dashboard View plugin:
* Set up a standard Dashboard View plugin.
* Customize portlets for different views.
* Configure notifications (e.g., email) for build status.

## Stage 2.5: Code Quality Tools

* Explore code quality tools like Sonar Cloud.
* Implement quality gates to maintain code quality standards.

# Stage 3: Building the Code & Configuring the Pipeline

## Stage 3.1: Task Pipelines

* Establish pipelines for various tasks related to your sample application (e.g., Java, C#).
* Define stages and actions within each pipeline.

## Stage 3.2: Deploy Application

* Deploy the application to a suitable web or application server.
* Ensure appropriate configuration for deployment.

## Stage 3.3: Build Pipeline for CI Lifecycle

* Set up a build pipeline that encompasses all stages of continuous integration.
* Include processes such as compilation, testing, and packaging within the pipeline.

# Stage 4: End-to-End Automation Of The Application Delivery Lifecycle

## Stage 4.1: Configuration Management Standardisation

## Stage 4.2: Docker Installation and Configuration

## Stage 4.3: Creation of CentOS Containers

# Stage 5: Cloud Provisioning and Configuration Management

## Stage 5.1: Resource Provisioning in a Cloud Environment

* Identify the cloud environment for resource provisioning.
* Determine the resources needed to support the sample application.

## Stage 5.2: Installing the Runtime Environment

* Install the necessary components required to run the sample application.
* Ensure compatibility and functionality with the chosen cloud environment.
* Verify successful installation and runtime environment configuration.

# Stage 6: Deploying Application (AWS, Azure, and Docker)

## Stage 6.1: Environment Selection

* Determine the target environment for deployment, considering options like cloud services (AWS, Azure, Google Cloud), container platforms (Docker, Kubernetes), or on-premises servers.

## Stage 6.2: Configuration Management Preparation

* Ensure that the configuration management tool has completed all necessary preparations for deployment.
* Verify that the application's configurations, dependencies, and environment variables are correctly set up and packaged.

## Stage 6.3a: Cloud Environment Deployment

* If deploying to a cloud environment:
  + Access the chosen cloud platform's management console.
  + Create or select the appropriate instance types, virtual machines, or containers for deployment.
  + Configure networking, security groups, access control policies, and firewall rules as needed.

## Stage 6.3b: Container-Based Deployment

* If deploying using containers:
  + Set up a container orchestration platform like Docker Swarm or Kubernetes.
  + Build container images for the sample application.
  + Push container images to a container registry like Docker Hub or a private registry.

## Stage 6.4: Deployment Configuration

* Define deployment configurations such as environment variables, secret management, and application scaling policies.
* Configure load balancers, auto-scaling groups, or service discovery mechanisms based on deployment requirements.

## Stage 6.5: Deployment Execution

* Initiate the deployment process, either manually or through automated deployment pipelines.
* Monitor the deployment progress and handle any errors or issues that arise during deployment.

## Stage 6.5: Post-Deployment Testing

* Conduct thorough testing to ensure that the application functions correctly in the new environment.
* Perform integration tests, end-to-end tests, and performance tests to validate the deployment's integrity.

## Stage 6.6: Monitoring and Maintenance

* Set up monitoring and logging tools to track the application's performance and health in the new environment.
* Implement alerts and notifications for critical events and performance degradation.
* Establish a maintenance schedule for routine updates, patches, and infrastructure optimizations.

# Stage 7: Monitoring Infrastructure and Applications

* Detail the necessity of continuous monitoring within the end-to-end automation process.
* Emphasise its critical role in ensuring service availability and application functionality.
* Highlight the significance of monitoring various aspects including cloud resources, application servers, and application performance.
* Stress the importance of early issue detection and timely resolution to prevent service disruptions.
* Explain how monitoring aids in optimizing resource utilization, identifying performance bottlenecks, and enhancing user experience.
* Illustrate the overarching goal of continuous monitoring: to increase services and application availability by proactively managing and addressing potential issues.

# Stage 8: Orchestrating Application Deployment

## Stage 8.1: Configure Build Jobs for Checkout and Execution:

* Set up build jobs to initiate the checkout of source code repositories and execute the build pipeline.

## Stage 8.2: Implement Compilation and Unit Test Execution:

* Include compilation tasks within the build jobs to compile the source code effectively.
* Integrate unit test execution to validate the functionality and integrity of the application.

## Stage 8.3: Provision Runtime Environment:

* Incorporate tasks to install the runtime environment, such as deploying Linux on Amazon EC2 instances, to support application execution.

## Stage 8.4: Configure Permissions in New Instances:

* Define configuration steps to configure permissions within the newly created instances.
* Ensure secure access and operation of the runtime environment and application.

## Stage 8.5: Automate Deployment Process:

* Automate the deployment process within the build jobs to streamline application rollout.
* Enable seamless deployment of the application to the provisioned runtime environment.