Pet Clinic

Automated Deployment Pipeline

Deirdre Lee

devops T00126583

Contents

[Stage 1: Code and Tools 3](#_Toc160545306)

[Stage 1.1: Summary of Application 3](#_Toc160545307)

[Stage 1.2: Tool Chain 3](#_Toc160545308)

[Stage 1.3: Flow Diagram 3](#_Toc160545309)

[Stage 2: Continuous Integration 3](#_Toc160545310)

[Stage 2.1: Install & Configure Continuous Integration 3](#_Toc160545311)

[Stage 2.2: Automate Build Process 3](#_Toc160545312)

[Stage 2.3: Configure Git Authentication 3](#_Toc160545313)

[Stage 2.4: Unit Test Execution 3](#_Toc160545314)

[Stage 2.5: Code Quality Tools 3](#_Toc160545315)

[Stage 3: Building the Code & Configuring the Pipeline 3](#_Toc160545316)

[Stage 3.1: Task Pipelines 3](#_Toc160545317)

[Stage 3.2: Deploy Application 4](#_Toc160545318)

[Stage 3.3: Build Pipeline for CI Lifecycle 4](#_Toc160545319)

[Stage 4: End-to-End Automation Of The Application Delivery Lifecycle 4](#_Toc160545320)

[Stage 4.1: Configuration Management Standardisation 4](#_Toc160545321)

[Stage 4.2: Docker Installation and Configuration 4](#_Toc160545322)

[Stage 4.3: Creation of CentOS Containers 4](#_Toc160545323)

[Stage 5: Cloud Provisioning and Configuration Management 4](#_Toc160545324)

[Stage 5.1: Resource Provisioning in a Cloud Environment 4](#_Toc160545325)

[Stage 5.2: Installing the Runtime Environment 4](#_Toc160545326)

[Stage 6: Deploying Application (AWS, Azure, and Docker) 4](#_Toc160545327)

[Stage 6.1: Environment Selection 4](#_Toc160545328)

[Stage 6.2: Configuration Management Preparation 4](#_Toc160545329)

[Stage 6.3a: Cloud Environment Deployment 5](#_Toc160545330)

[Stage 6.3b: Container-Based Deployment 5](#_Toc160545331)

[Stage 6.4: Deployment Configuration 5](#_Toc160545332)

[Stage 6.5: Deployment Execution 5](#_Toc160545333)

[Stage 6.5: Post-Deployment Testing 5](#_Toc160545334)

[Stage 6.6: Monitoring and Maintenance 5](#_Toc160545335)

[Stage 7: Monitoring Infrastructure and Applications 5](#_Toc160545336)

[Stage 8: Orchestrating Application Deployment 6](#_Toc160545337)

[Stage 8.1: Configure Build Jobs for Checkout and Execution: 6](#_Toc160545338)

[Stage 8.2: Implement Compilation and Unit Test Execution: 6](#_Toc160545339)

[Stage 8.3: Provision Runtime Environment: 6](#_Toc160545340)

[Stage 8.4: Configure Permissions in New Instances: 6](#_Toc160545341)

[Stage 8.5: Automate Deployment Process: 6](#_Toc160545342)

# Stage 1: Code and Tools

## Stage 1.1: Summary of Application

Pet clinic is an application developed using the Spring Boot framework, which can be built using either Maven or Gradle, with the help of included wrapper scripts. (spring-projects, n.d.) Its web interface consists of a homepage and a navigation bar with three options: the ‘owners/find’ endpoint, where you can add an owner using a form or search for one using the last name; A ‘/vets’ endpoint, listing all vets along with options to view the results as XML or JSON; and an ‘/oups’ endpoint demonstrating the page shown when an exception occurs.

Many components from the Spring stack were used in the construction of this project, including Spring MCV, which provides a model-view-controller architecture, and Spring Data JPA, facilitating interaction with the associated database. (Singh Raina & Giraldo, 2022). Thymeleaf serves as the template engine, used for server-side rendering. Additionally, the in-memory H2 database is used by default to store data during runtime. The Junit framework is used for the unit testing.

The application was sourced from the Spring Projects repository. (spring-projects, n.d.)

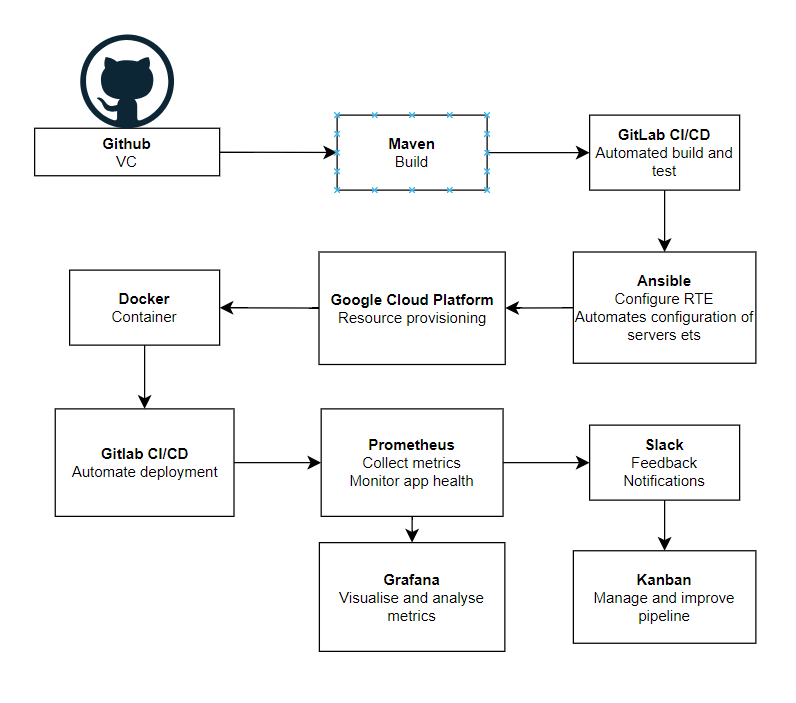
## Stage 1.2: Tool Chain

* Add relevant DevOps technologies - Continuously update the list as the semester progresses.

|  |  |
| --- | --- |
|  | TOOL CHAIN |
| Source Code Repositories | GitHub |
| Build Tools | Maven |
| Continuous Integration | Jenkins\*  Travis CI  Circle CI  GitLab CI/CD\* |
| Configuration Management to Set Runtime Environment | Ansible\*  Puppet  Chef |
| Resource Provisioning ToolsCloudContainers | AWS  Azure  Google Cloud Platform  Docker  Google Kubernetes Engine (GKE) |
| Continuous Delivery & Deployment | Circle CI  Travis CI  GitLab CI/CD\* |
| Continuous Monitoring | Prometheus\*  Grafana\* |
| Continuous Feedback | Slack\*  Email |
| Continuous Improvement | Kanban\*  Jira |
| Continuous Integration | Jenkins\* Travis CI Circle CI |

## Stage 1.3: Flow Diagram

* Highlight the technologies to be implemented at each stage.
* Temporary based on plan for pipeline – once tool/service has been implemented will update/add icon



# Stage 2: Continuous Integration

## Stage 2.1: Install & Configure Continuous Integration

As I had Jenkins and the GitHub Integration plugin installed previously it was not necessary to install them again. I forked the Pet Clinic application from spring-projects/spring-petclinic repository, storing it in a public GitHub repository, and set up continuous integration using Jenkins. I configured a Jenkins job to automatically build and test the application whenever changes are pushed to the repository using a hook trigger, as shown in Fig 1 and 2.

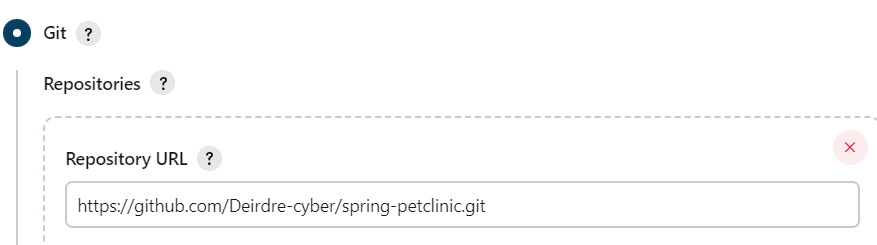


Fig 1 Link repository

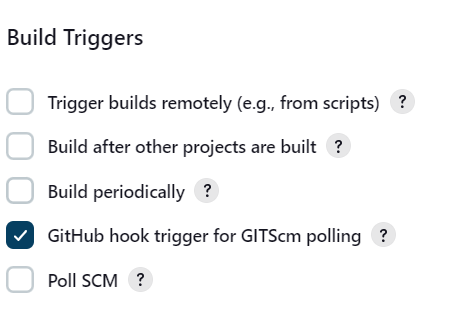


Fig 2 Build trigger

Next, I defined the build steps to execute the Maven goal ‘clean install’ to ensure a clean build lifecycle by deleting the target directory, depicted in Fig 3.

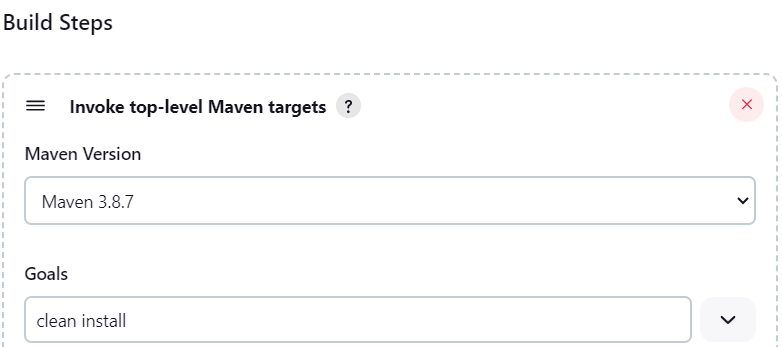
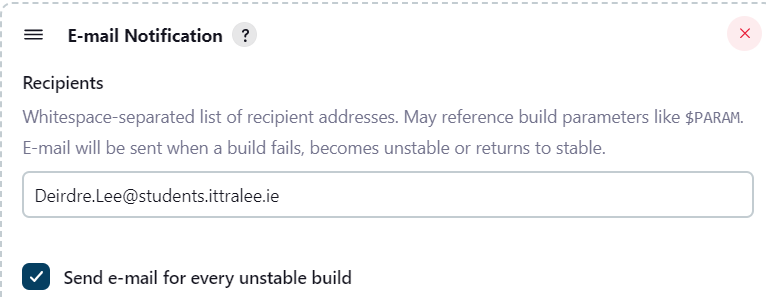


Fig 3 Build goal

Finally, in addition to generating a surefire test result report, I added an additional post-build action of sending an email to alert of an unstable build.



## Stage 2.2: Automate Build Process

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

I have chosen to use the build tool Maven as part of the automation process of this project.

Create build script in pom…

Branches for build environments…

## 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.