**DevOps - Automation**

**Continuous Integration and Continuous Delivery (CI/CD)**

**A PROJECT REPORT**

*Submitted in partial fulfilment for the award of the degree*

*of*

**Master of Science**

***in***

**Information Technology**

*by*

**SANKET SURESH PETHKAR**

**(14MIN2879)**

*Under the guidance of*

**Prof. Sandeep Patil**

**VIT**

****



**School of Information Technology and Engineering**

June, 2018

****

**DECLARATION BY THE CANDIDATE**

I hereby declare that the thesis entitled **“DevOps – Automation (CI/CD Pipeline)”** submitted by me to Vellore Institute of Technology, Vellore, in partial fulfilment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** is a record of bonafide project work carried out by me under the supervision of **Sandeep Patil**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

**Place**:

**Date**: **Signature of the Candidate**

**School of Information Technology and Engineering**

****

**BONAFIDE CERTIFICATE**

This is to certify that the project work entitled “**DevOps – Automation (CI/CD Pipeline)” by Sanket Suresh Pethkar (14MIN2879),** to Vellore Institute of Technology, Vellore, in partial fulfilment of the requirement for the award of the degree of **Master of Technology** in **Information Technology**, is a project bonafide work carried out by him/her under my supervision. The project fulfils the requirement as per the regulations of this Institute and in my opinion, meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this Institute or any other Institute or University.

**Prof. Sandip Patil**

**Internal Supervisor**

**VIT**

**Internal Examiner(s) External Examiner(s)**

**Contents**

[1. Introduction vi](#_gjdgxs)

[1.1 Software Configuration and Release management vi](#_30j0zll)

[1.2 CI/CD vi](#_1fob9te)

[2. Issues in existing system vi](#_3znysh7)

[3. Proposed Solution vi](#_2et92p0)

[3.1. OBJECTIVES: vii](#_tyjcwt)

[4. Software and hardware specifications vii](#_3dy6vkm)

[5 Architecture Design viii](#_1t3h5sf)

[5.1 ARCHITECTURE DIAGRAM viii](#_4d34og8)

[5.1.1 CI/CD Pipeline Workflow viii](#_2s8eyo1)

[5.1.2 DevOps Architecture ix](#_17dp8vu)

[5.2 Use Case Diagrams x](#_3rdcrjn)

[5.2.1When to Use: Use Cases Diagrams x](#_26in1rg)

[5.3 Class Diagrams xi](#_lnxbz9)

[5.3.1 When to Use: Class Diagrams xii](#_35nkun2)

[5.4 Sequence diagrams xii](#_1ksv4uv)

[6. Implementation xiv](#_44sinio)

6.1 Continuous Integration [xv](#_z337ya)

6.2 Continuous Delivery [xvii](#_3whwml4)

[6.3 Microservices xvii](#_2bn6wsx)

[6.3 Infrastructure as Code xvii](#_qsh70q)

[6.4 Monitoring and Logging xviii](#_3as4poj)

[6.5 Communication and Collaboration xviii](#_1pxezwc)

[7 Module Description xix](#_49x2ik5)

[7.1 GitHub xix](#_2p2csry)

[7.2 Jenkins xix](#_147n2zr)

[7.3 Nexus xx](#_3o7alnk)

8. Reports [xx](#_ihv636)

[9. Screenshots. xxvi](#_32hioqz)

[10. Abbreviations xxvi](#_1hmsyys)

11. REFERRENCES xxvi

**1. Introduction**

## 1.1Software Configuration and Release management

## Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products. SCM helps in identifying individual elements and configurations, tracking changes, and version selection, control, and baselining.

## Release management is the process of managing, planning, scheduling and controlling software build through different stages and environments, including testing and deploying software releases.

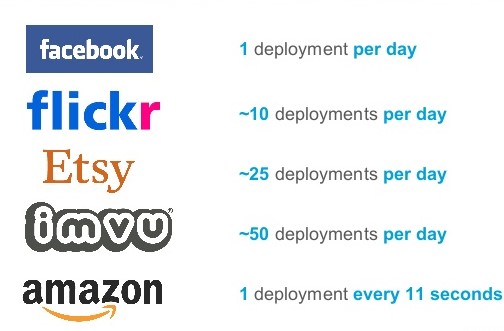
## 1.2CI/CD

**Continuous Delivery (CD)** is about automating Software Release Management (automated infrastructure provisioning, automated build, automated deploy and automated testing)

**Continuous integration (CI)** systems provide automation of the software build and validation process driven in a continuous way by running a configured sequence of operations every time a software change is checked into the source code management repository.

## 2. Issues in existing system

## In the current fast-moving IT world where everyone is moving to Agile, the number of deployments per unit time is increased so much that it is difficult to manage the infrastructure provisioning, build, deploy and testing manually. Manual work will need large number of resources, more time and there will be chance of errors and issues due to bulk work. To overcome all these challenges automation is required.

**Some Examples**:

**3. Proposed Solution**

**Implementing a CI/CD Pipeline for SDLC**

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

**DevOps Practices**

The following are DevOps best practices:

* Continuous Integration
* Continuous Delivery
* Micro services
* Infrastructure as Code
* Monitoring and Logging
* Communication and Collaboration

## 3.1. OBJECTIVES:

Below are DevOps goals:

* Release on demand
* Eliminate technical debt and unplanned work
* Fail smart/fast/safe
* Look "outside-in"
* Measure feature value

## 4. Software and hardware specifications

**Hardware:**

Processor : Intel Core i5 or equivalent processor

RAM : Base machine – 8 GB or more

VM – 8 GB or more

ROM : Base machine – 1 TB or more

VM – 500 GB or more

**Software:**

Operating system : Base machine – Windows 7 or later

Front End Software : JAVA, HTML

Back End Software : SQL Server 2005

* Jenkins
* Nexus
* GitHub
* Ansible / Chef
* Docker
* MySQL, PostgreSQL
* Nginx, Jetty Server

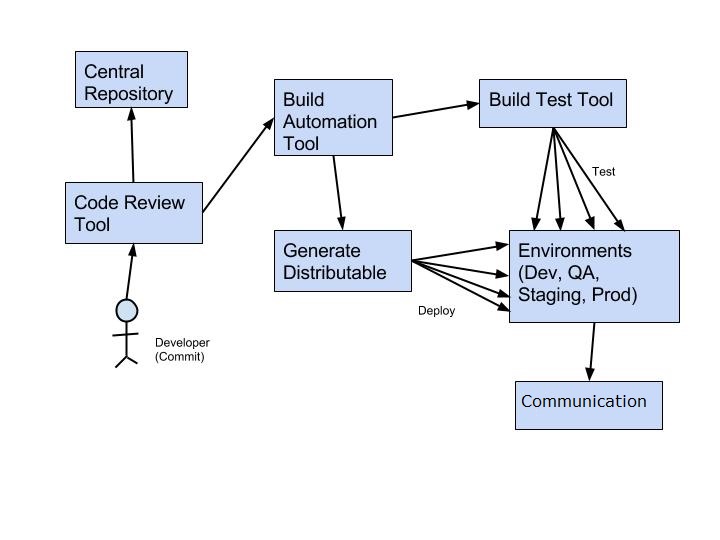
## 

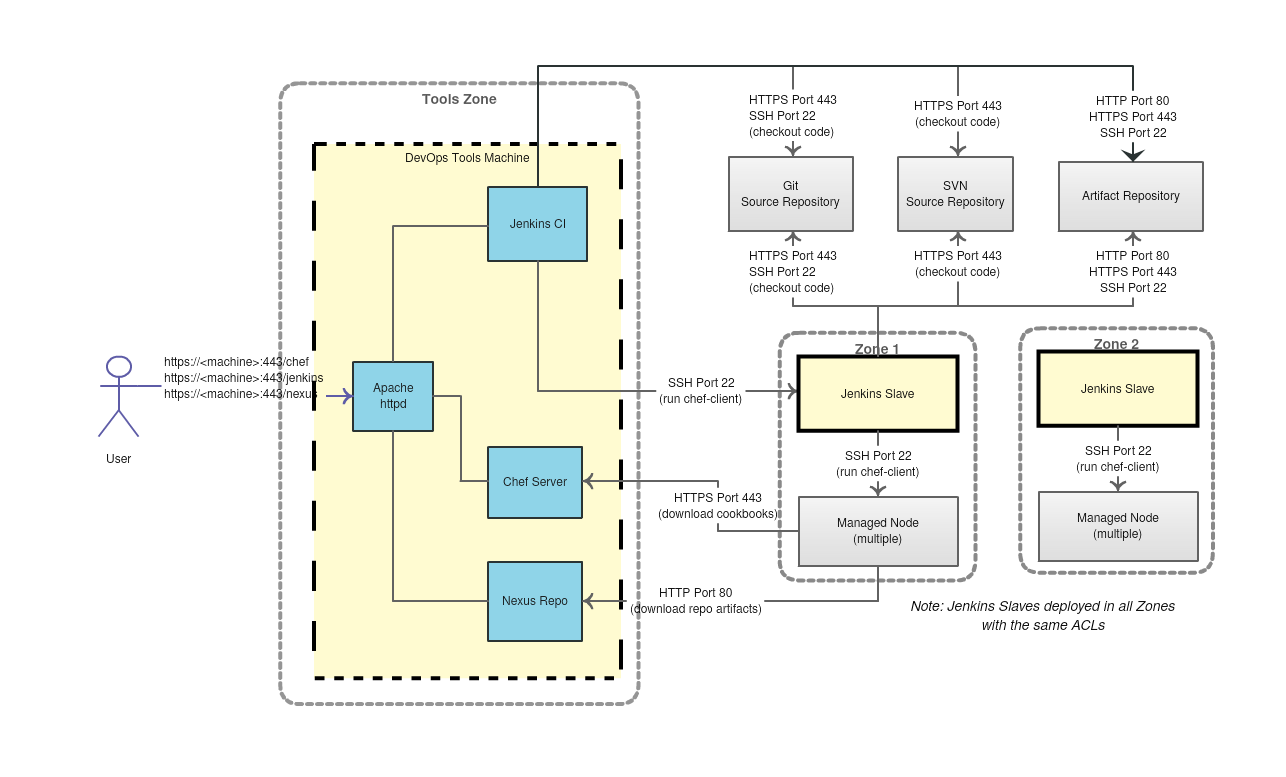
## 

## 5 Architecture Design

## 5.1 ARCHITECTURE DIAGRAM

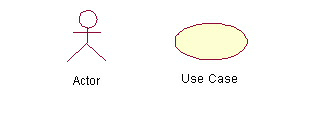
## C:\Users\SA292082\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CICDWorkflow.png5.1.1 CI/CD Pipeline Workflow

**5.1.2 DevOps Architecture**



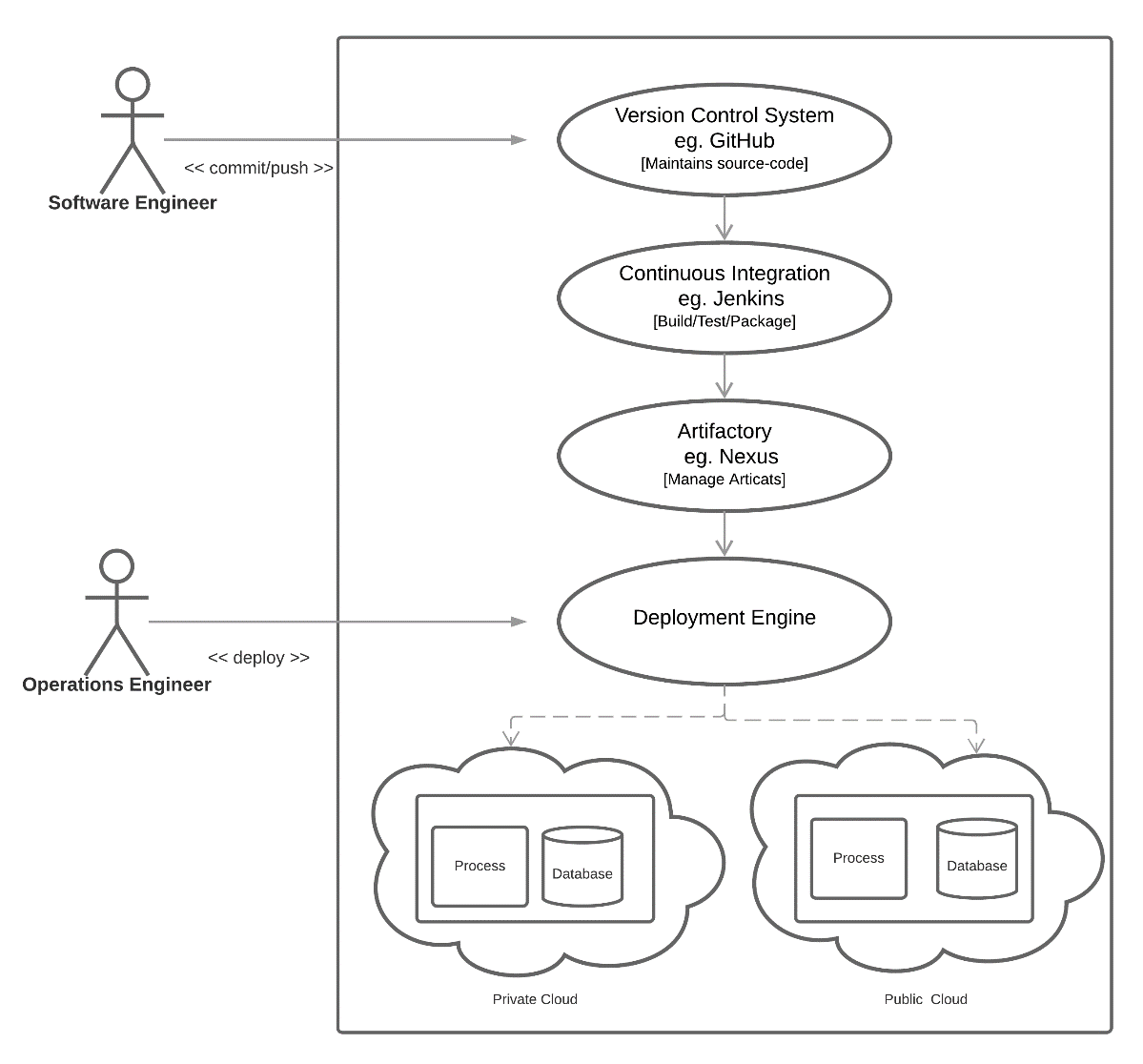
## 5.2 Use Case Diagrams

A use case is a set of scenarios that describing an interaction between a user and a system.  A use case diagram displays the relationship among actors and use cases.  The two main components of a use case diagram are use cases and actors.



## 5.2.1When to Use: Use Cases Diagrams

Use cases are used in almost every project.  They are helpful in exposing requirements and planning the project. During the initial stage of a project most use cases should be defined, but as the project continues more might become visible.

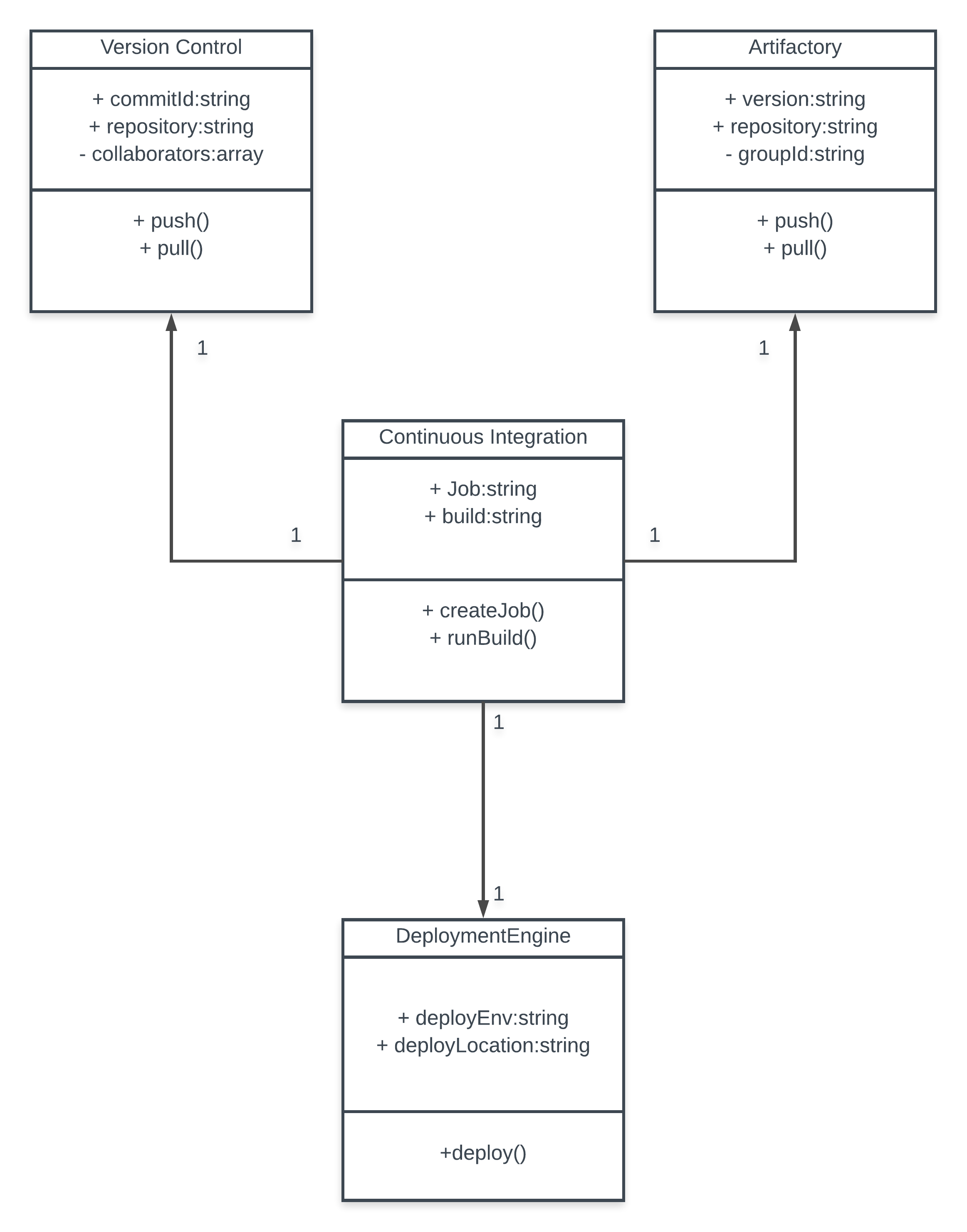


## 5.3 Class Diagrams

Class diagrams are widely used to describe the types of objects in a system and their relationships.  Class diagrams model class structure and contents using design elements such as classes, packages and objects.  Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design.  This example is only meant as an introduction to the UML and class diagrams.  If you would like to learn more see the Resources page for more detailed resources on UML.

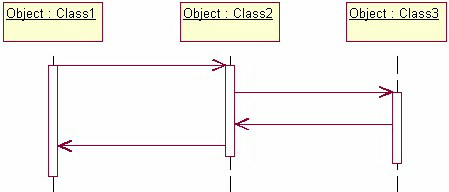
## 5.3.1 When to Use: Class Diagrams

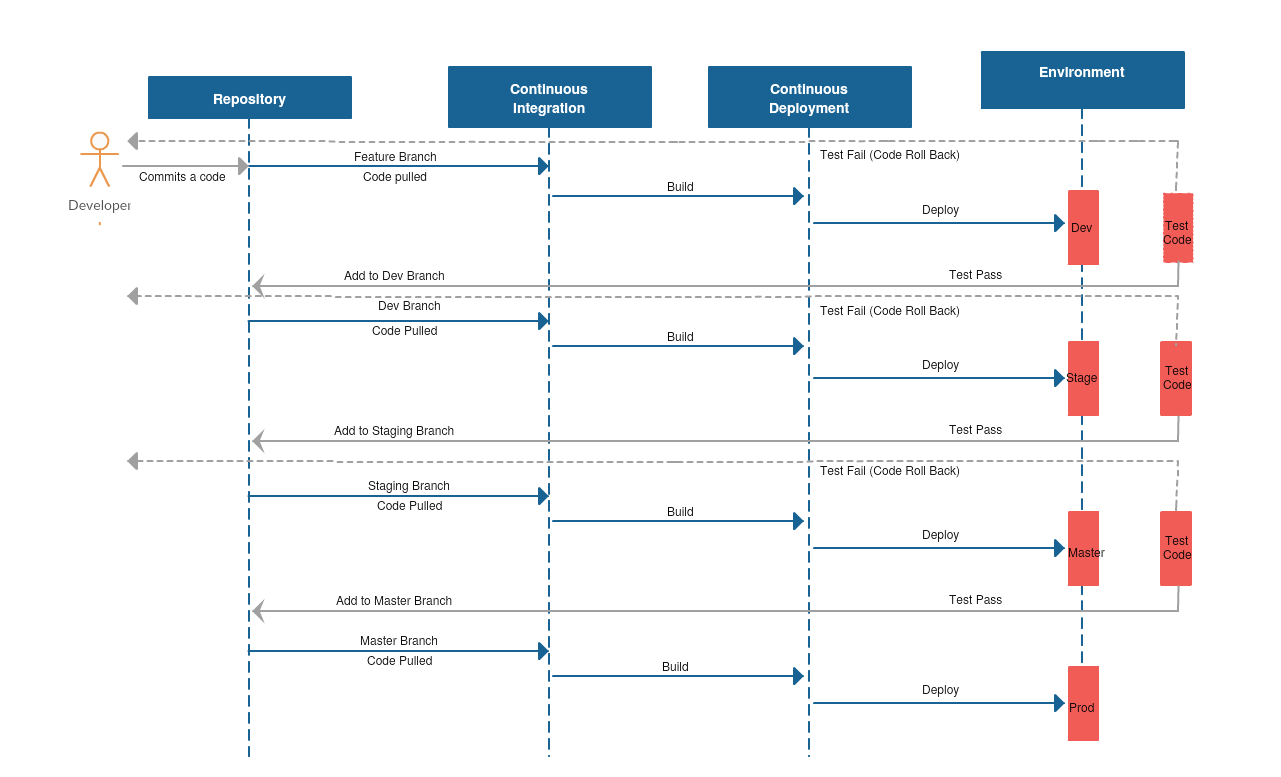
Class diagrams are used in nearly all Object-Oriented software designs. Use them to describe the Classes of the system and their relationships to each other.



## 5.4 Sequence diagrams:

Sequence diagrams demonstrate the behaviour of objects in a use case by describing the objects and the messages they pass.  The diagrams are read left to right and descending.  The example below shows an object of class 1 start the behaviour by sending a message to an object of class 2.  Messages pass between the different objects until the object of class 1 receives the final message.





## 6. Implementation

## The increasing adoption of DevOps by most of the organizations across the globe clearly indicates its potential as a key enabler to achieving scale. Implementation of DevOps practices helps an organization deliver faster, better, high-quality and reliable software relying on the culture of cooperation and collaboration among all functions of an organization. It calls for fundamental cultural changes and modification of legacy programming practices.  Here are the core DevOps best practices that help an organization achieve the goals of effective communication and collaboration, smoother operations and bug-free code. Here are the ten key recommendations for successful DevOps implementation:

## 6.1 Continuous Integration

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

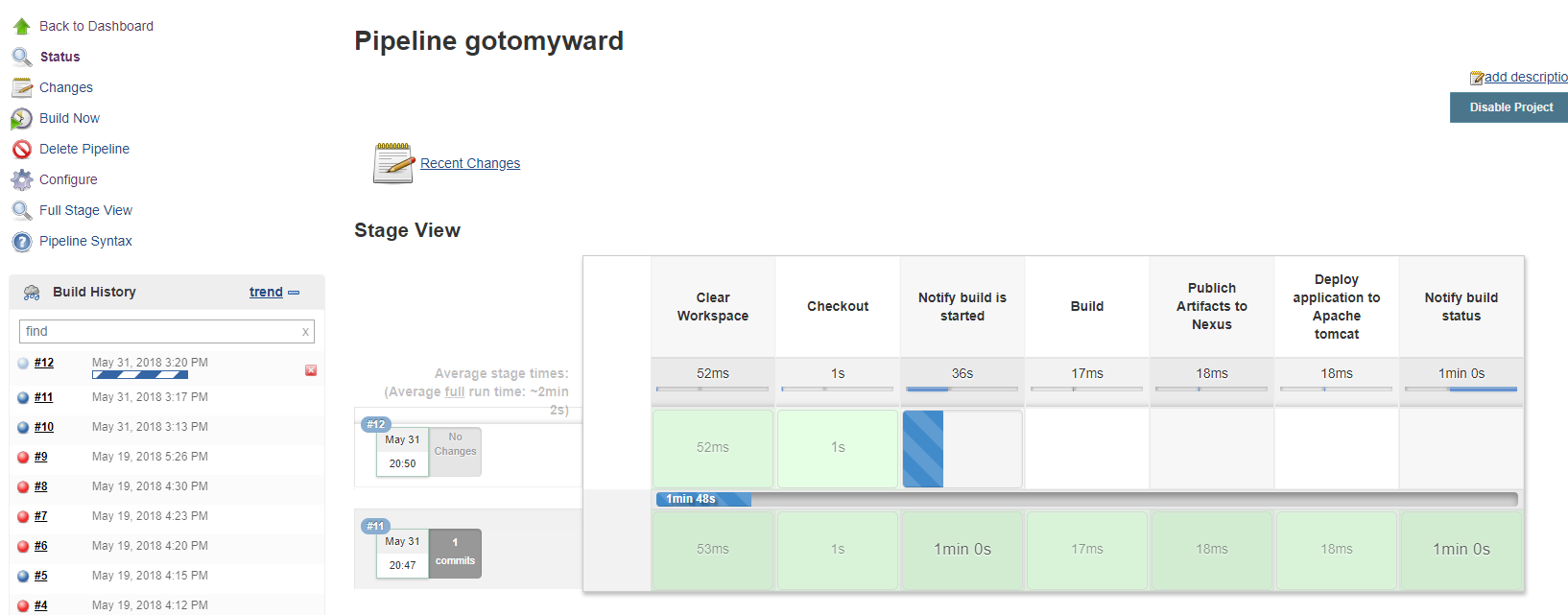
We are using Jenkins as Continuous Integration tool which performs orchestration between all the tools like GitHub – SCM, Gradle – build tool, Nexus – Artifactory, etc. It controls and manages the flow of build pipeline.

## 6.2 Continuous Delivery

## Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

## We are using Jenkins for deploying applications on the application servers. It will copy the application to apache tomcat server by unzipping the artifacts wherever required.

**Jenkins CI/CD Pipeline**



## 6.3 Microservices

## The microservices architecture is a design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through a well-defined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API). Microservices are built around business capabilities; each service is scoped to a single purpose. You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services.

We are using docker for setting up various tools like Nexus and clustering Nexus with HA configuration. Docker containers are pulled and application is setup. For multiple applications we are using docker compose.

## 6.3 Infrastructure as Code

## Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. The cloud’s API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources. Thus, engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code. Because they are defined by code, infrastructure and servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways.

Ansible is configuration management tool and is used to install, upgrade and configure all the tools. We are creating Ansible playbooks in YML format for the required configuration will setup the infrastructure.

**Configuration Management**

Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more. The use of code makes configuration changes repeatable and standardized. It frees developers and systems administrators from manually configuring operating systems, system applications, or server software.

## 6.4 Monitoring and Logging

Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product’s end user. By capturing, categorizing, and then analyzing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes. Active monitoring becomes increasingly important as services must be available 24/7 and as application and infrastructure update frequency increases. Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services.

## 6.5 Communication and Collaboration

Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps. The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations. Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis. This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects.

## 7 Module Description

## 7.1 GitHub

|  |  |  |
| --- | --- | --- |
| Steps | Steps Description | Output |
| Step 1 | Login to GitHub | Login is successful |
| Step 2 | Go to Repository if exist. | Repository is opened or created. |
| Step 3 | Clone Repository to local | All the details should be entered successfully. Code should be committed. |

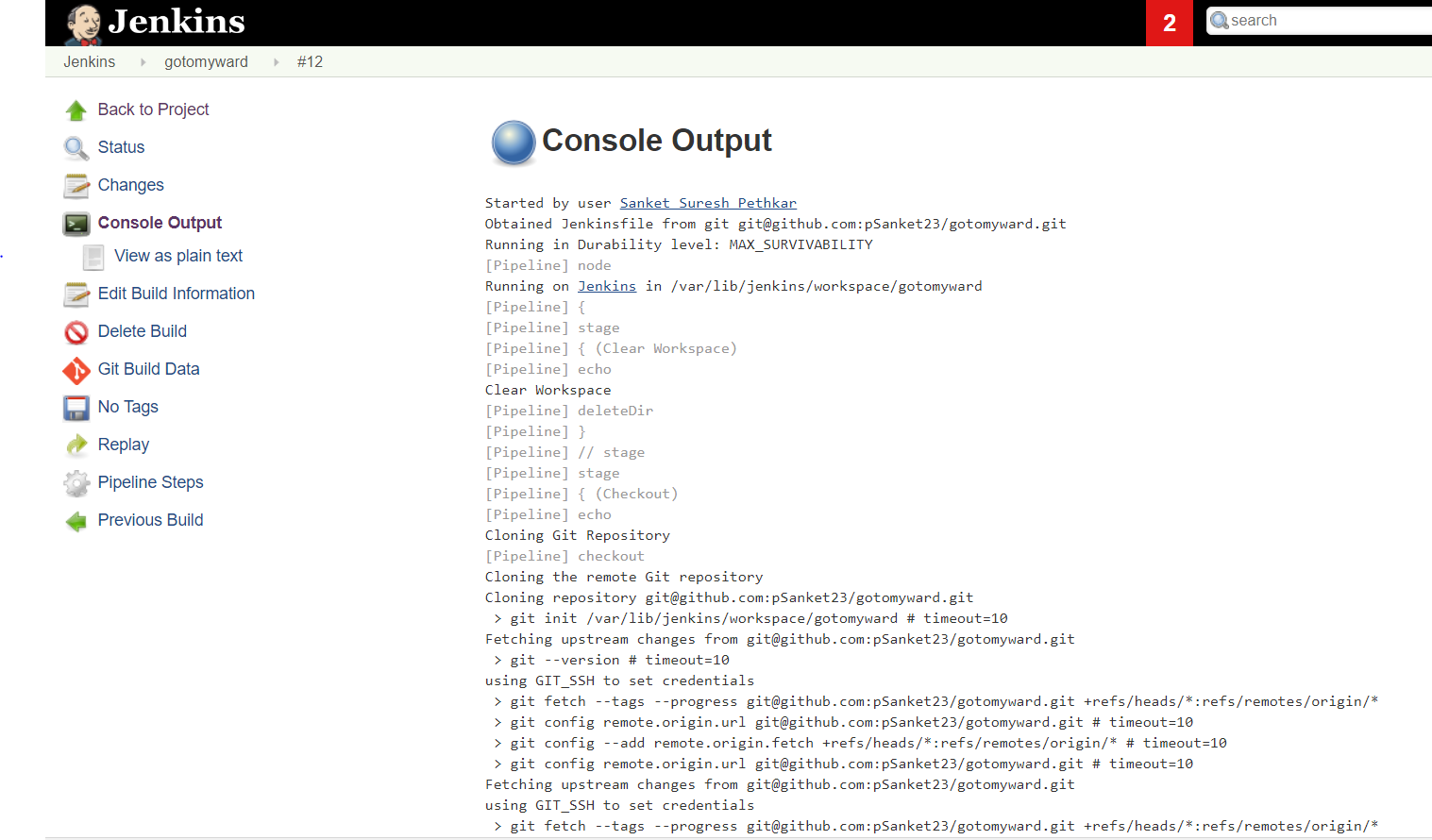
## 7.2 Jenkins

|  |  |  |
| --- | --- | --- |
| Step 1 | Login to Jenkins | Login is successful |
| Step 2 | Go to Jenkins 🡪 Job | Job pipeline is displayed. |
| Step 3 | Click on build now | Build is triggered. |
| Step 4 | Click on build number 🡪 Console Output | The build logs are displayed. |

## 7.3 Nexus

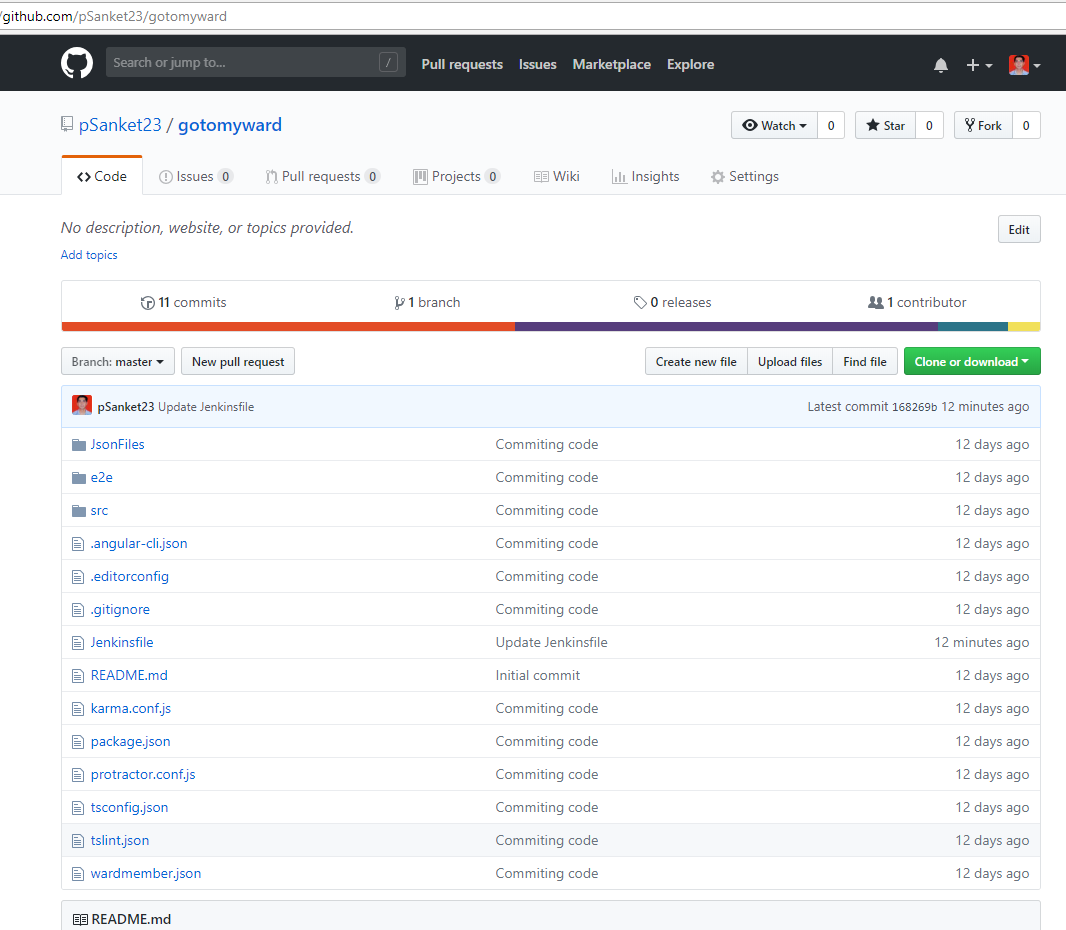
|  |  |  |
| --- | --- | --- |
| Step 1 | Login to Nexus | Login is successful |
| Step 2 | Go to Browse 🡪 Repository | Repositories are displayed |
| Step 3 | Click on repo name | All versions of the artifacts are displayed. |

## 8. Reports

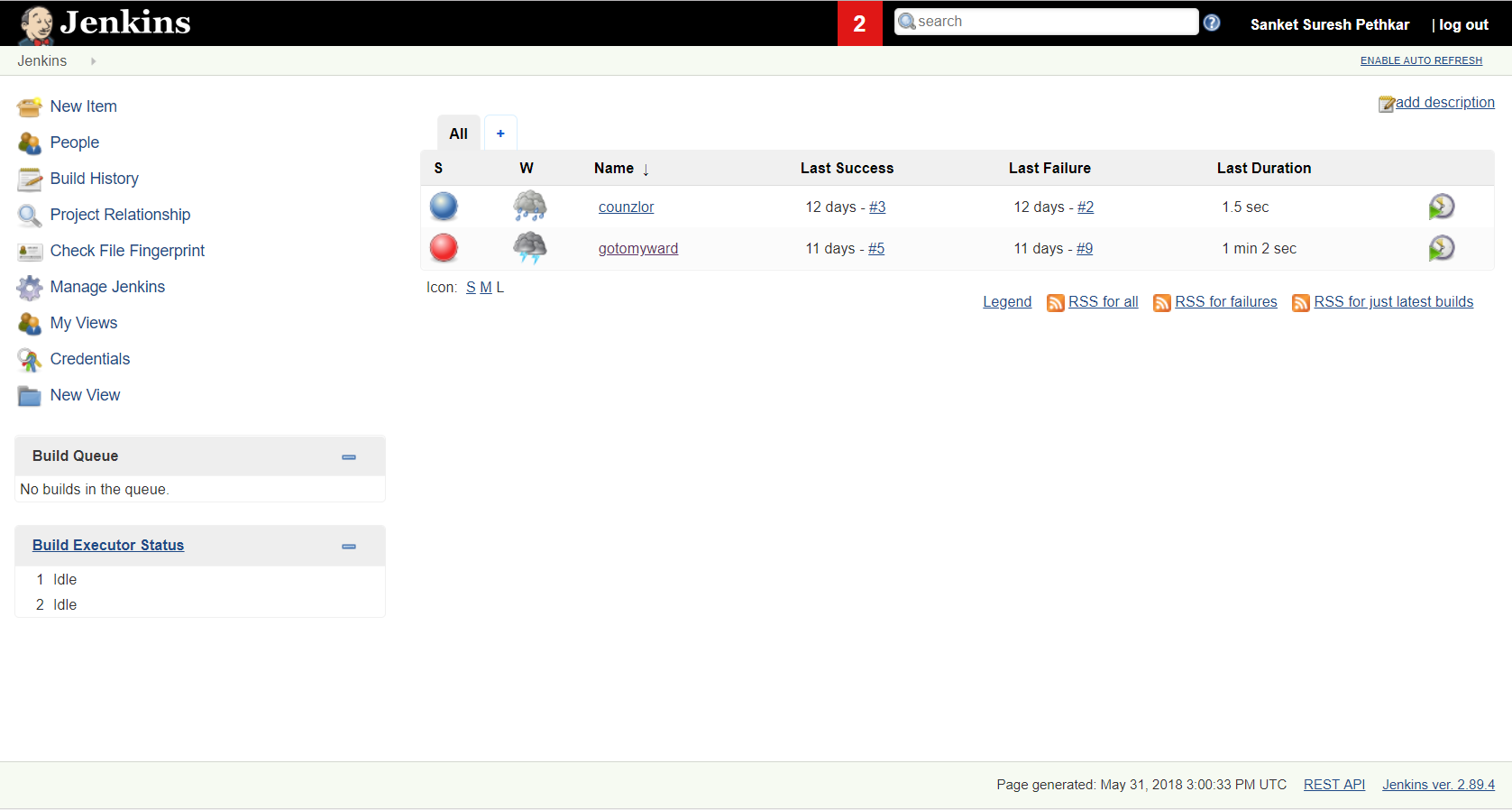


## 9. Screenshots.

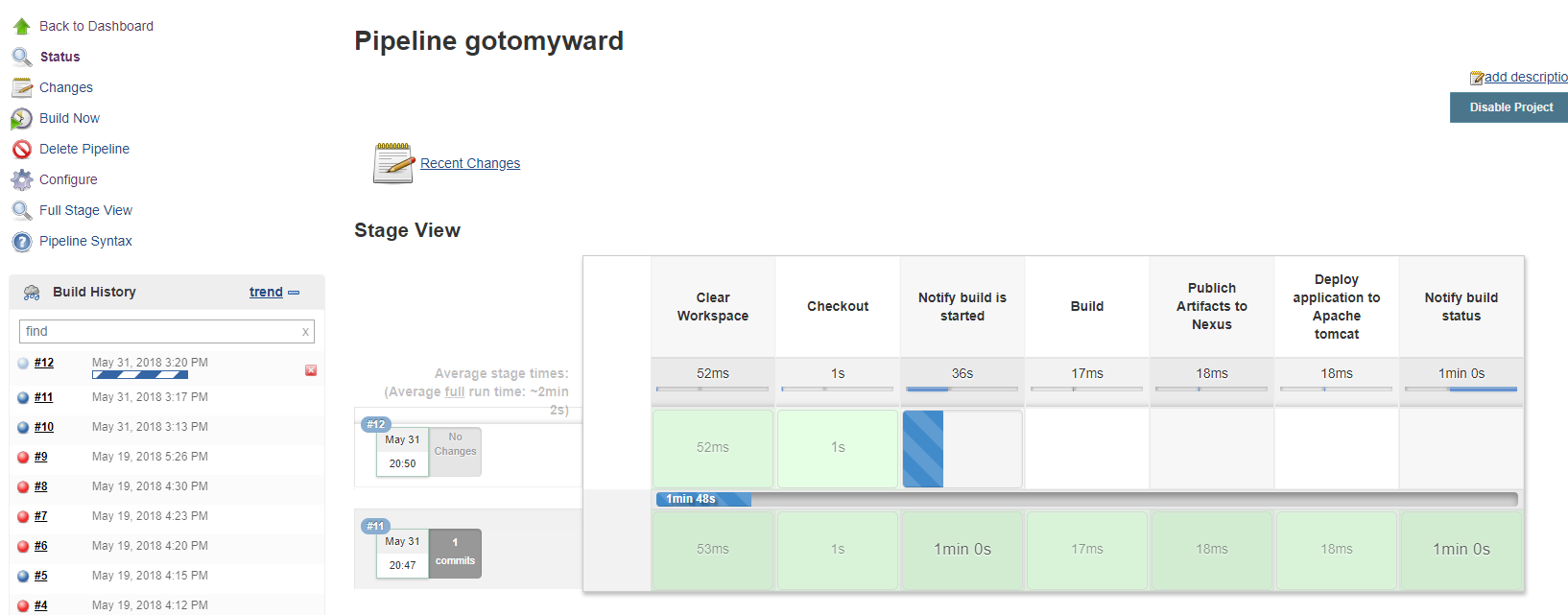
GitHub Repository.



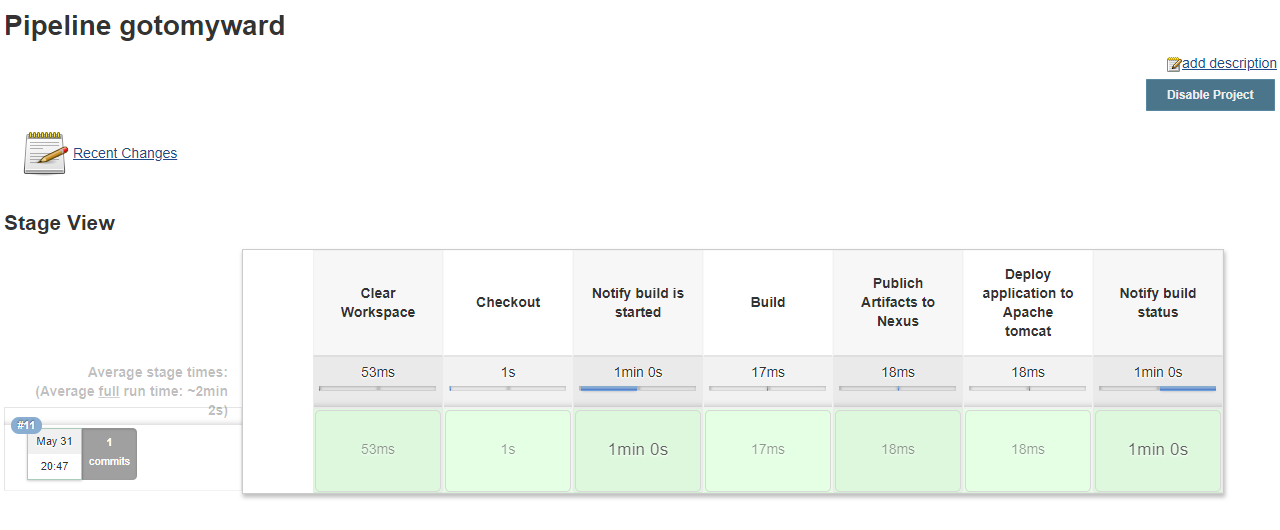
## Jenkins Homepage



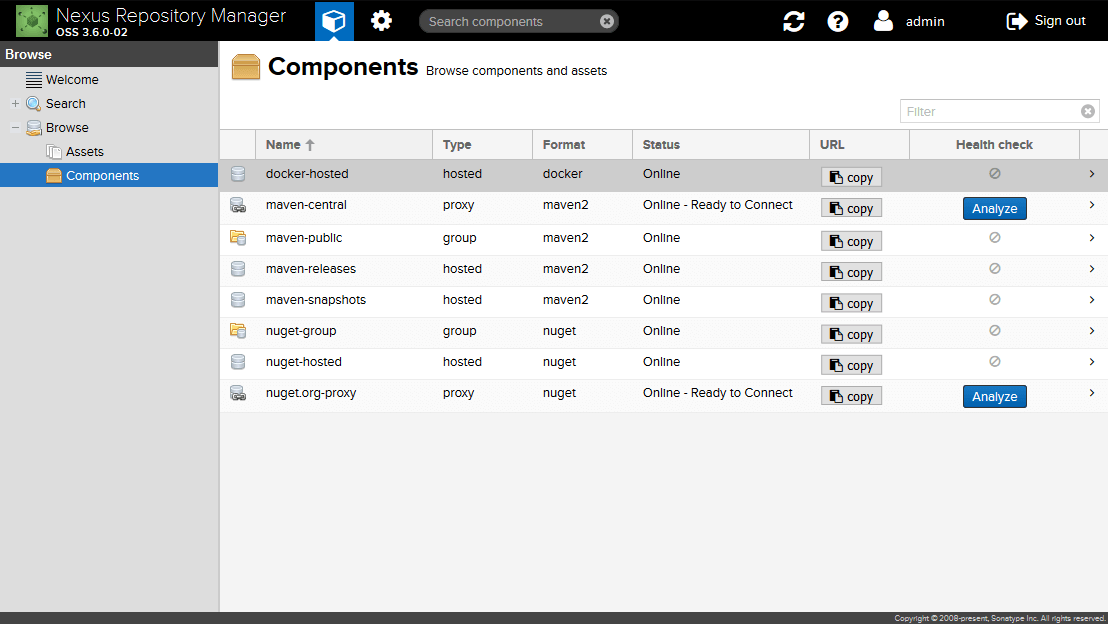
## Jenkins Pipeline



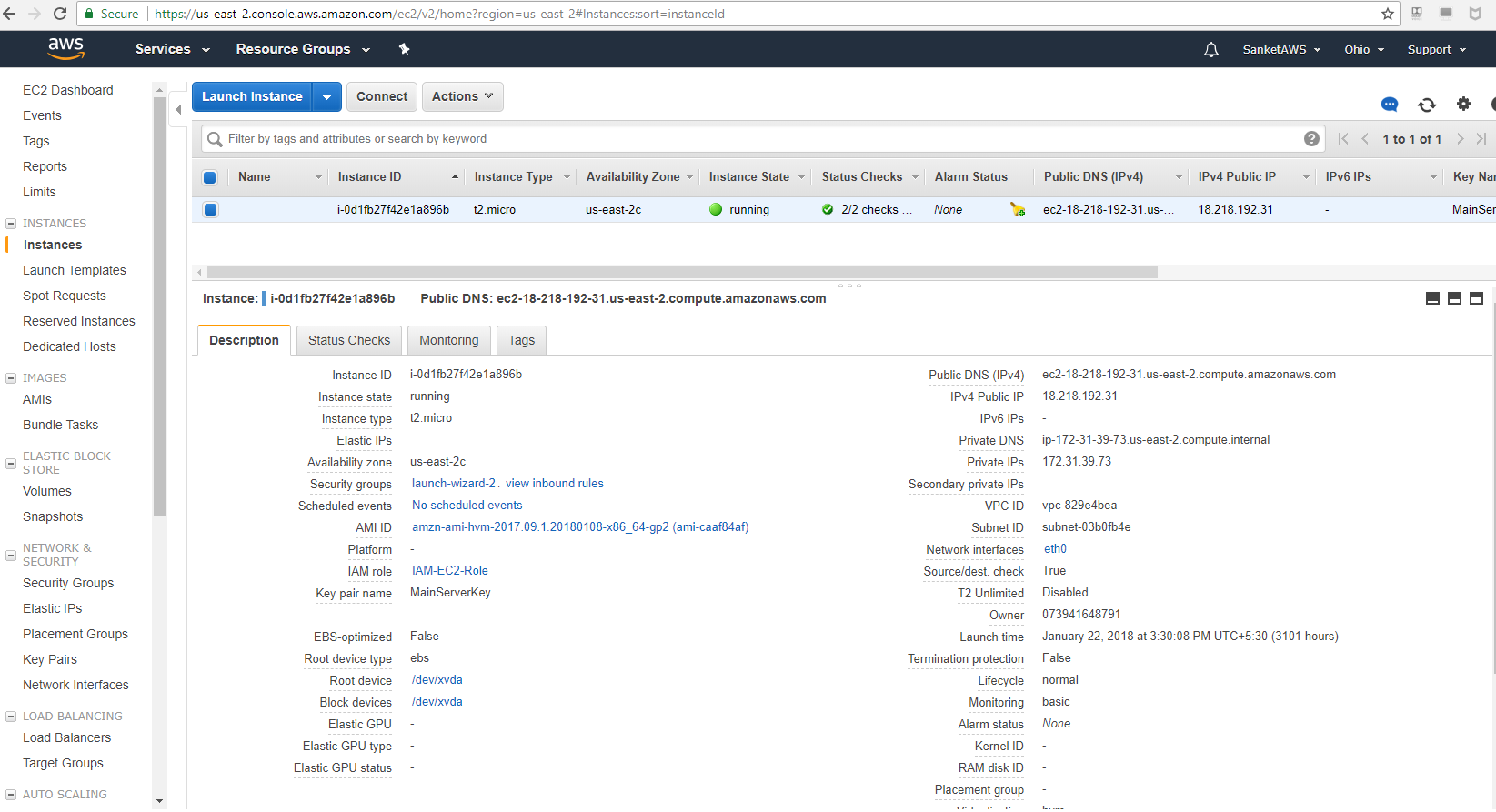
Successful Job



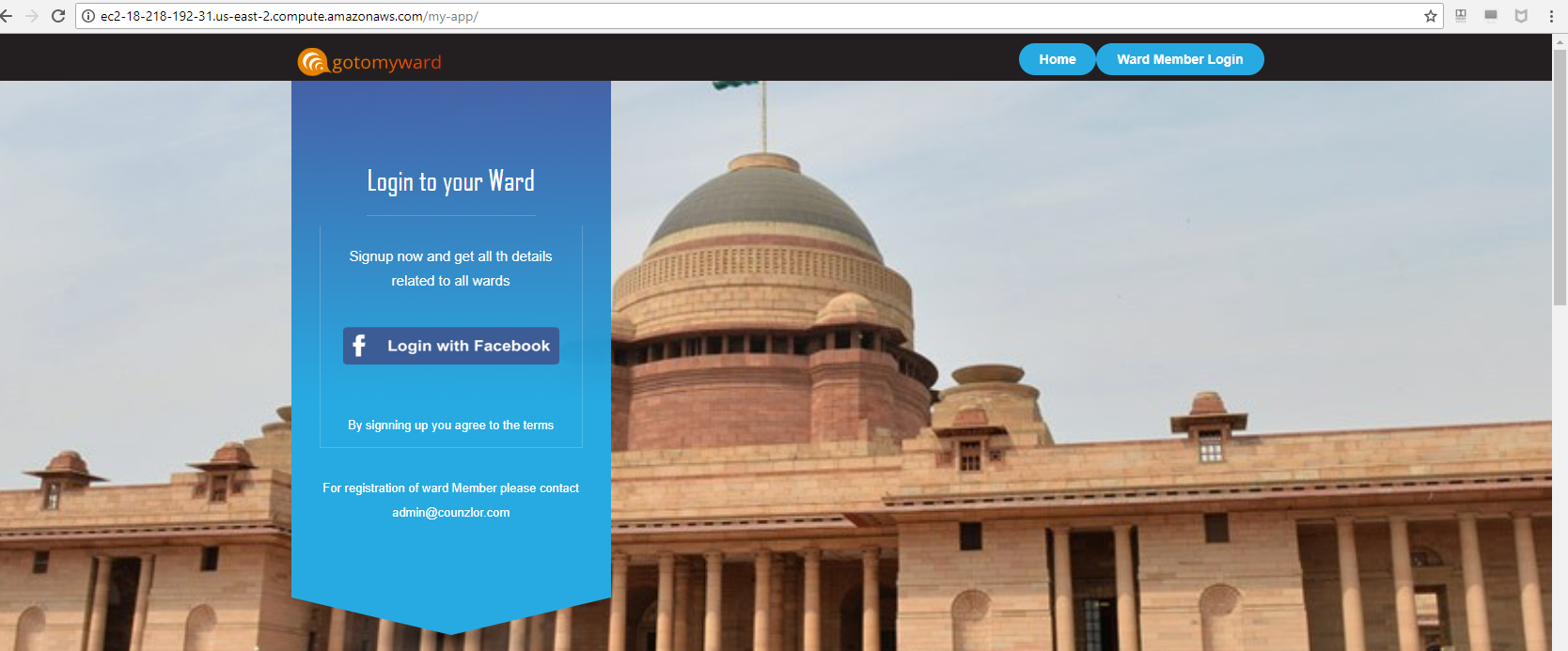
Nexus



AWS EC2 instance



Application Deployed to AWS EC2 server



## 10. Abbreviations

CI – Continuous Integration

CD – Continuous Delivery

HA – High Availability.

YML – Yet another mark-up language.

AWS – Amazon Web Services

EC2 – Elastic Cloud Compute

## 11. REFERRENCES

URL

<https://github.com>

<https://www.tutorialspoint.com/devops_tutorials.htm>

<https://jenkins.io/doc/>

<http://docs.ansible.com/>

<https://books.sonatype.com/nexus-book>

<https://docs.docker.com/>