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

<u>BONAFIDE CERTIFICATE</u>

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

1.1 Software Configuration and Release management vi

1.2 CI/CD vi

2. Issues in existing system vi

3. Proposed Solution vi

3.1. OBJECTIVES: vii

4. Software and hardware specifications vii

5 Architecture Design viii

5.1 ARCHITECTURE DIAGRAM viii

5.1.1 CI/CD Pipeline Workflow viii

5.1.2 DevOps Architecture ix

5.2 Use Case Diagrams x

5.2.1When to Use: Use Cases Diagrams x

5.3 Class Diagrams xi

5.3.1 When to Use: Class Diagrams xii

5.4 Sequence diagrams: xii

6. Implementation xiv

6.1 Continuous Integration xv

6.2 Continuous Delivery xvii

6.3 Microservices xvii

6.3 Infrastructure as Code xvii

6.4 Monitoring and Logging xviii

6.5 Communication and Collaboration xviii

7 Module Description xix

7.1 GitHub xix

7.2 Jenkins xix

7.3 Nexus xx

8. Reports xx

9. Screenshots. xxvi

10. Abbreviations xxvi

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

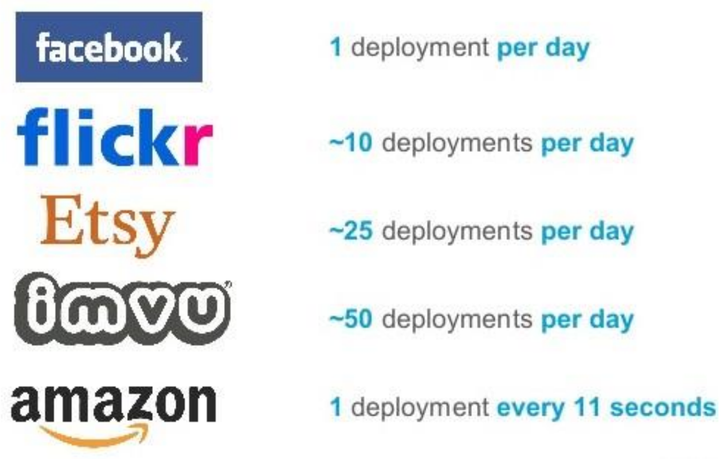
Continuous Delivery (CD) is about automating Software Release Management (automated infrastructure provisioning, automated build, automated deploy and automated testing)

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:



facebook	1 deployment per day
flickr	~10 deployments per day
Etsy	~25 deployments per day
imvu	~50 deployments per day
amazon	1 deployment every 11 seconds

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:

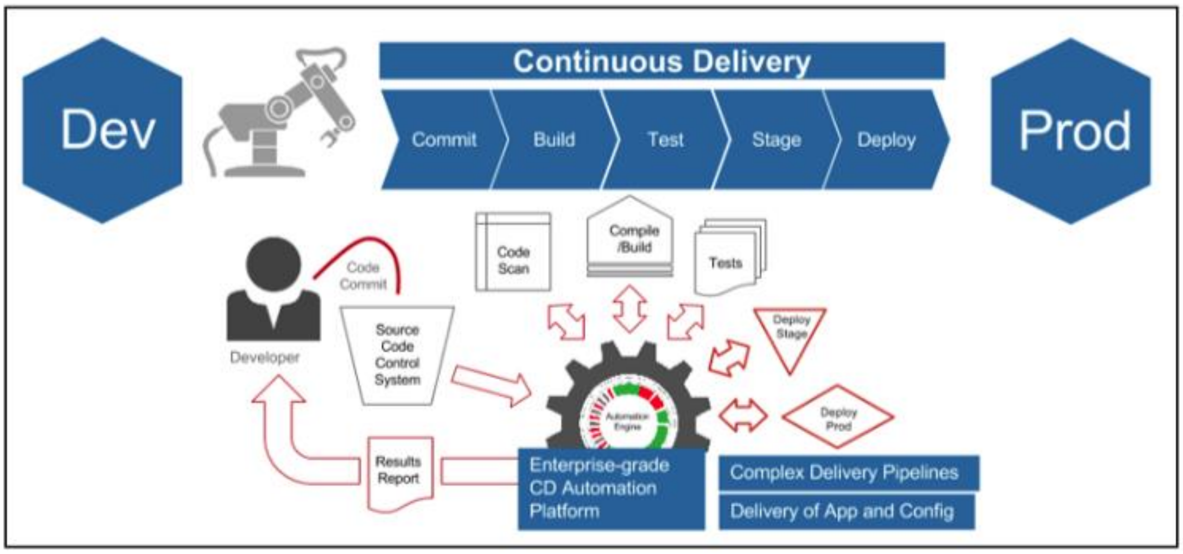
Operating system	:	Base machine – Windows 7 or later
Front End Software	:	JAVA, HTML
Back End Software	:	SQL Server 2005

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

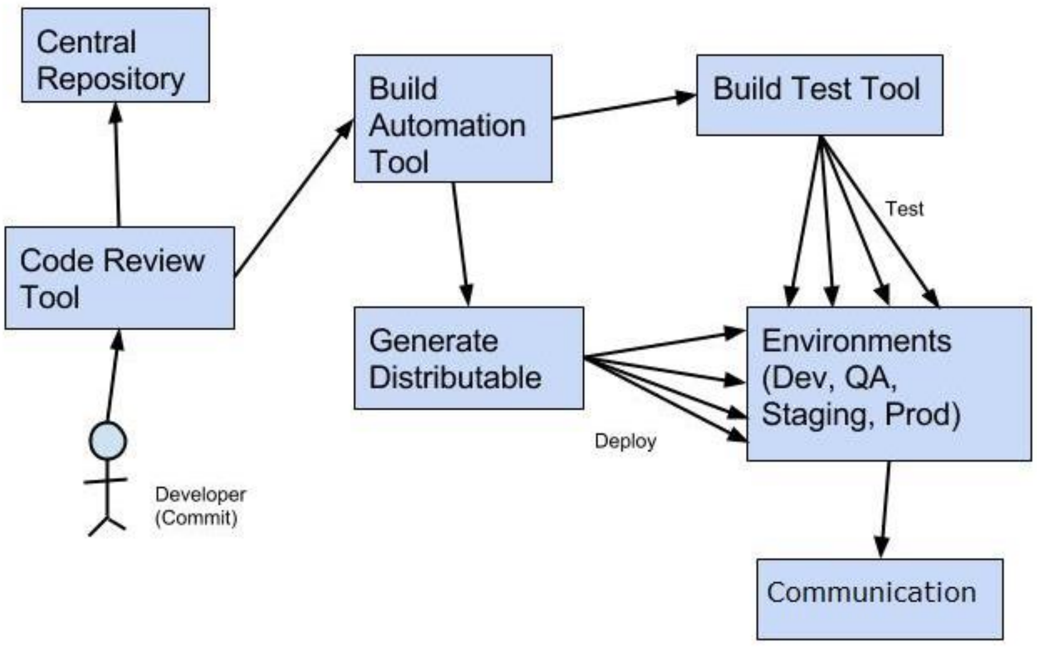
5 Architecture Design

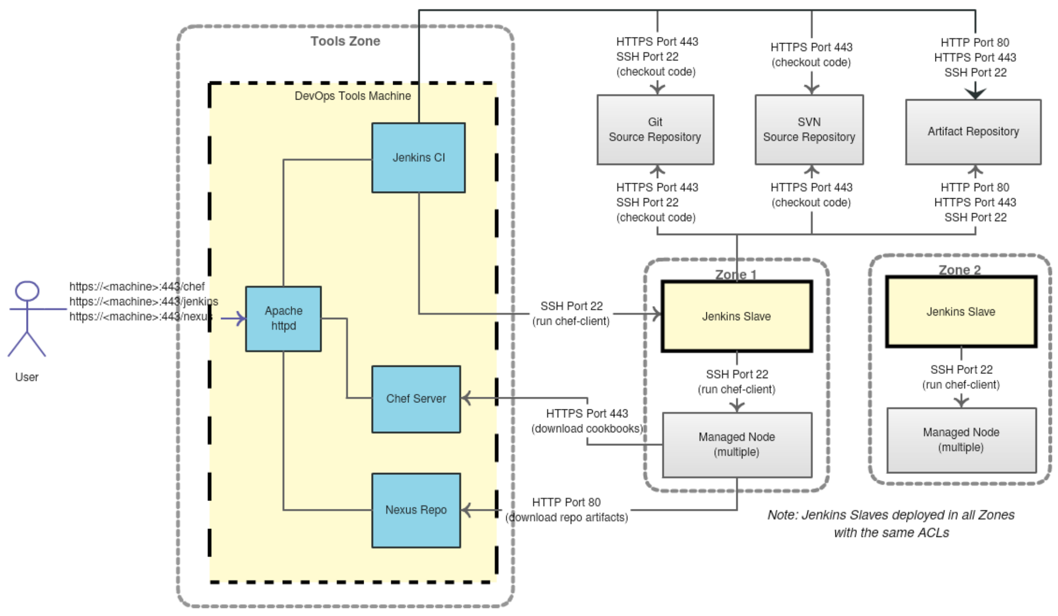
5.1 ARCHITECTURE DIAGRAM

5.1.1 CI/CD Pipeline Workflow



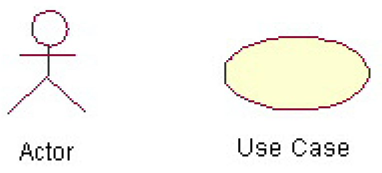
5.1.2 DevOps Architecture





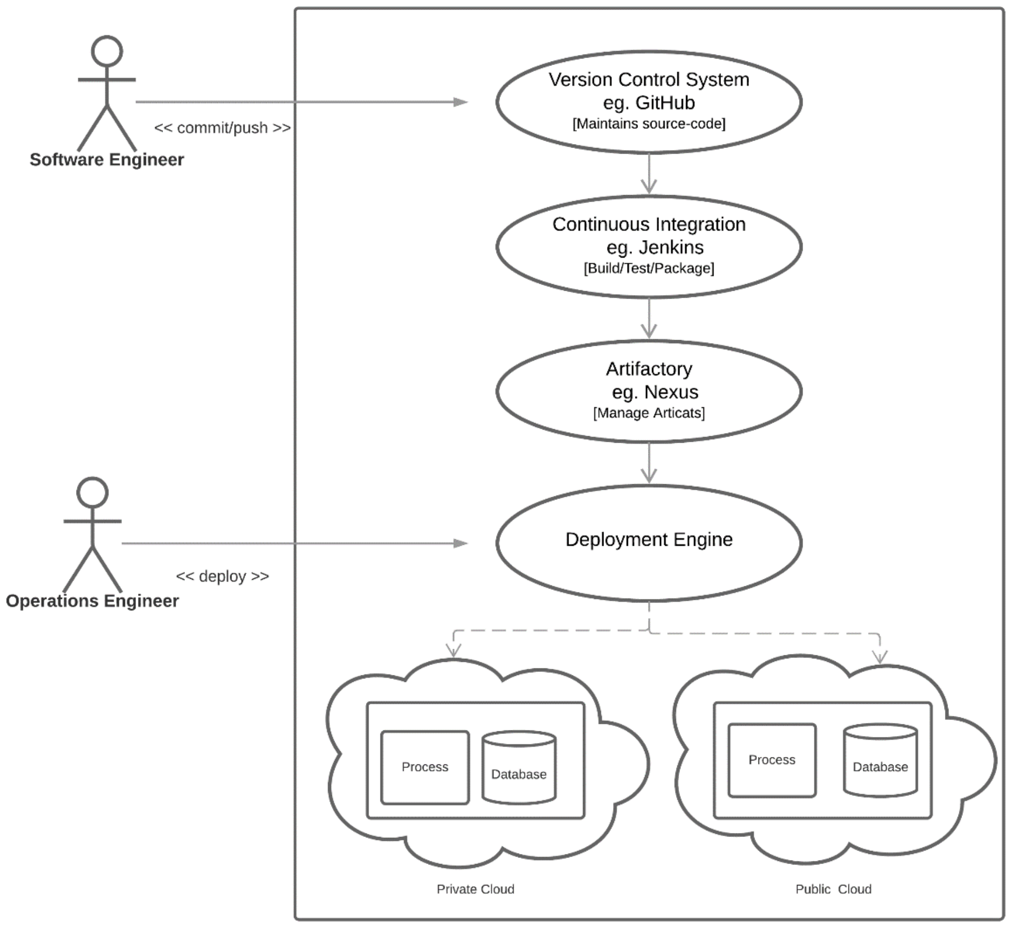
5.2 Use Case Diagrams

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.

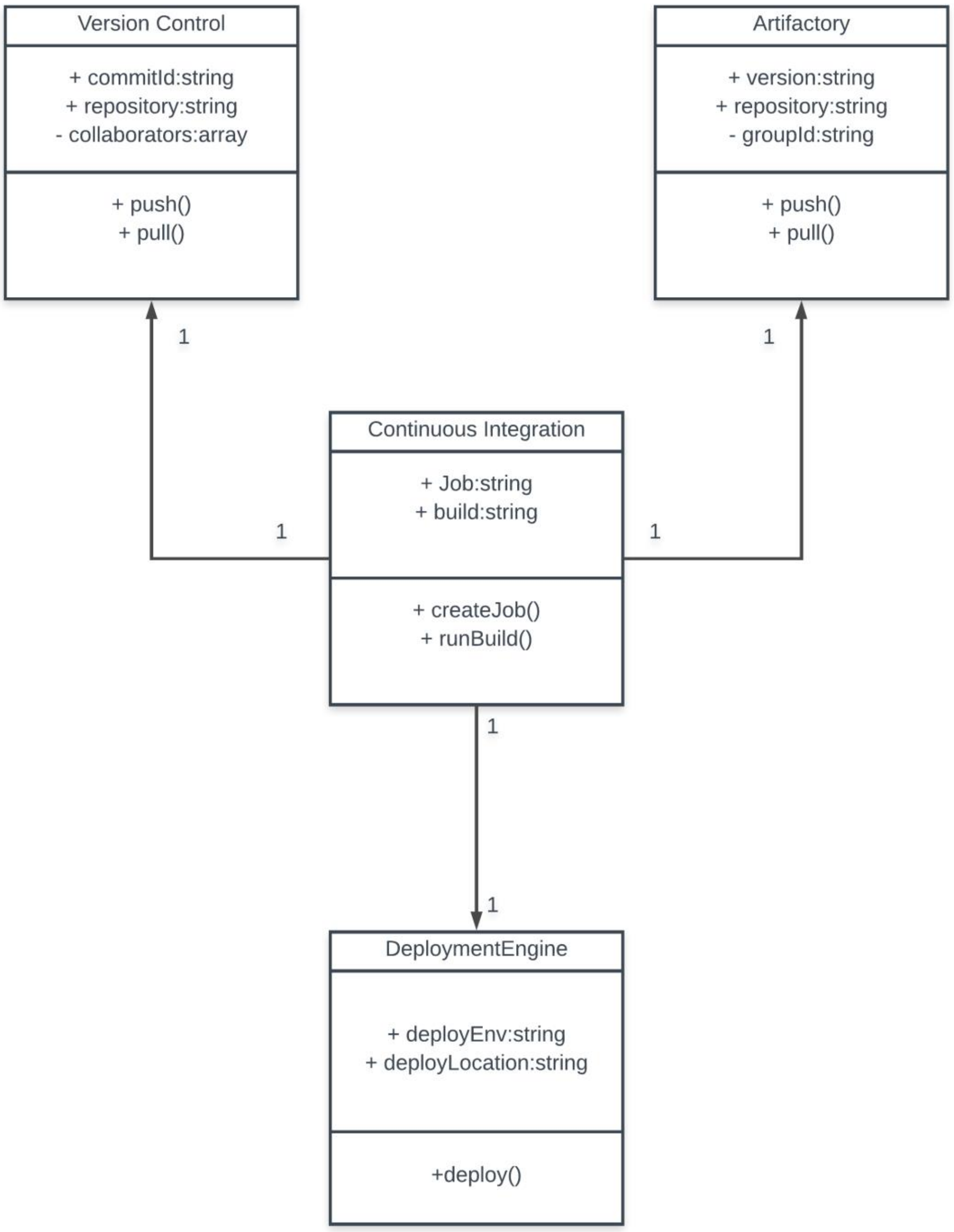


5.3 Class Diagrams

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.

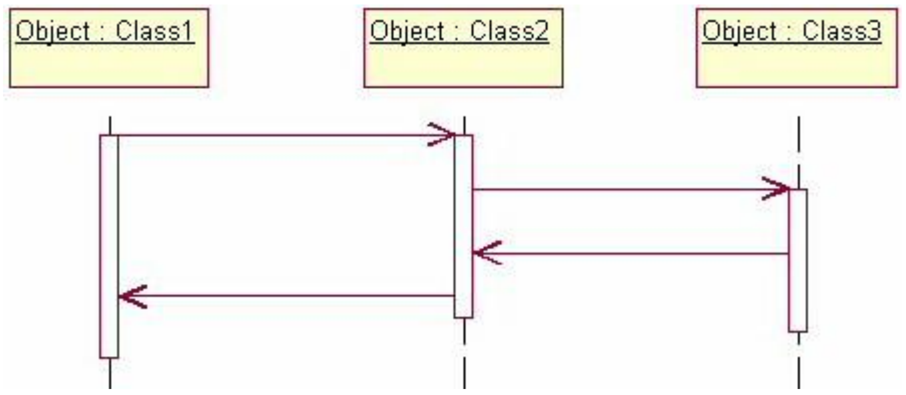
5.3.1 When to Use: Class Diagrams

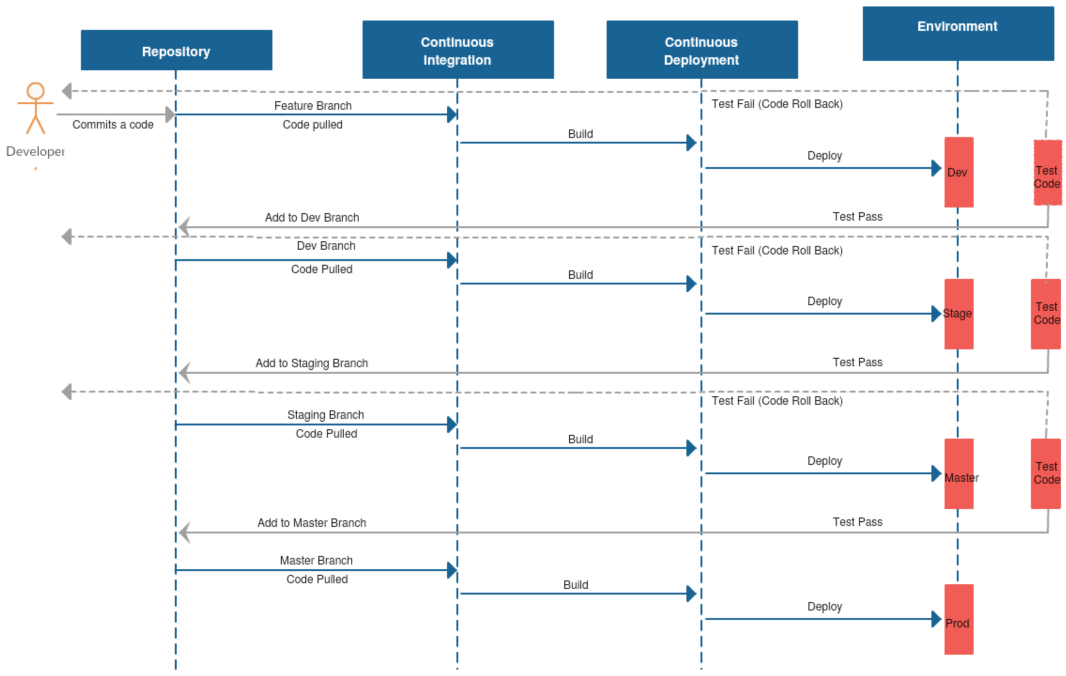
Class diagrams are used in nearly all Object-Oriented software designs. Use them to describe the Classes of the system and their relationships to each other.


Version Control
+ commitId:string
+ repository:string
- collaborators:array
+ push()
+ pull()
Artifactory
+ version:string
+ repository:string
- groupId:string
+ push()
+ pull()
Continuous Integration
+ Job:string
+ build:string
+ createJob()
+ runBuild()
DeploymentEngine
+ deployEnv:string
+ deployLocation:string
+deploy()
1
1
1
1
1
1

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.

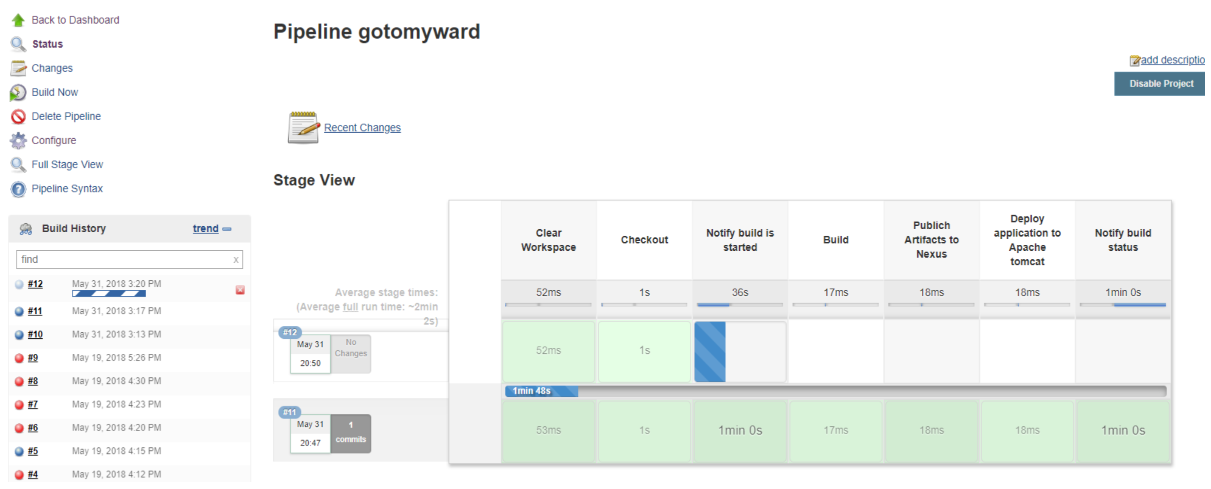
We are using Jenkins as Continuous Integration tool which performs orchestration between all the tools like GitHub – SCM, Gradle – build tool, Nexus – Artifactory, etc. It controls and manages the flow of build pipeline.

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



Pipeline gotomyward

Build History trend

#12	May 31, 2018 3:20 PM
#11	May 31, 2018 3:17 PM
#10	May 31, 2018 3:13 PM
#9	May 19, 2018 5:26 PM
#8	May 19, 2018 4:30 PM
#7	May 19, 2018 4:23 PM
#6	May 19, 2018 4:20 PM
#5	May 19, 2018 4:15 PM
#4	May 19, 2018 4:12 PM

Stage View

	Clear Workspace	Checkout	Notify build is started	Build	Publich Artifacts to Nexus	Deploy application to Apache tomcat	Notify build status
Average stage times: (Average full run time: ~2min 2s)	52ms	1s	36s	17ms	18ms	18ms	1min 0s
#12 May 31 20:50 No Changes	52ms	1s					
#11 May 31 20:47 1 commits	53ms	1s	1min 0s	17ms	18ms	18ms	1min 0s

1min 48s

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.

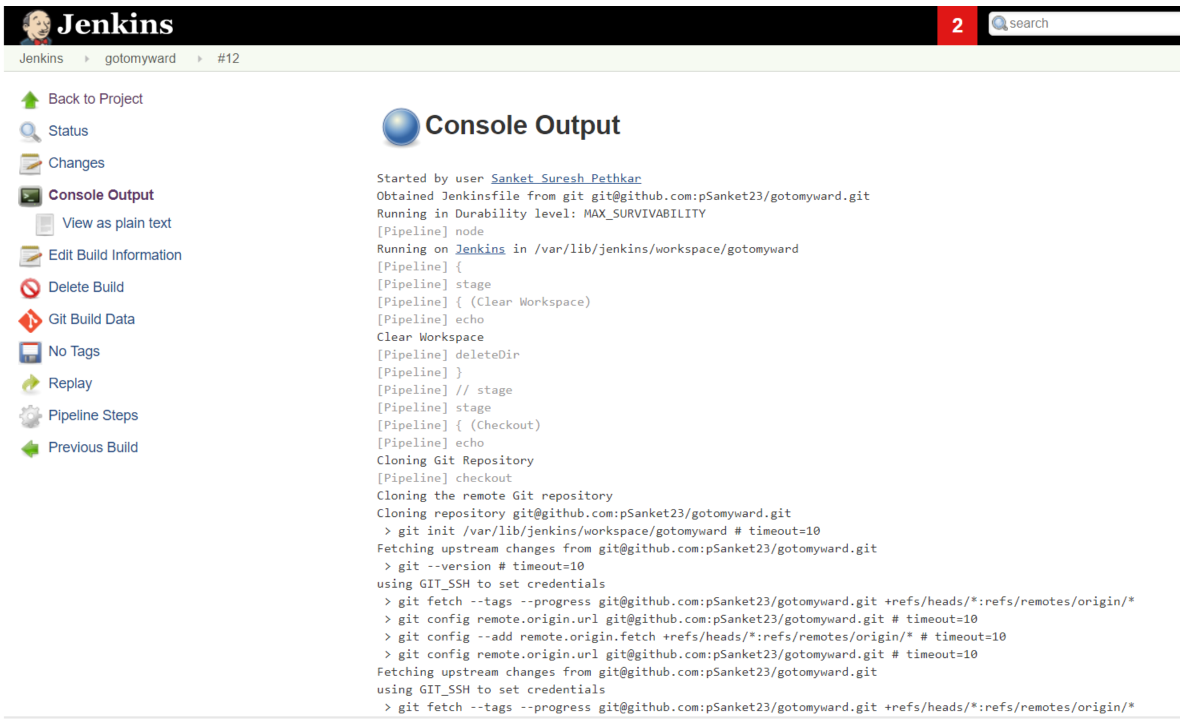
7.2 Jenkins

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

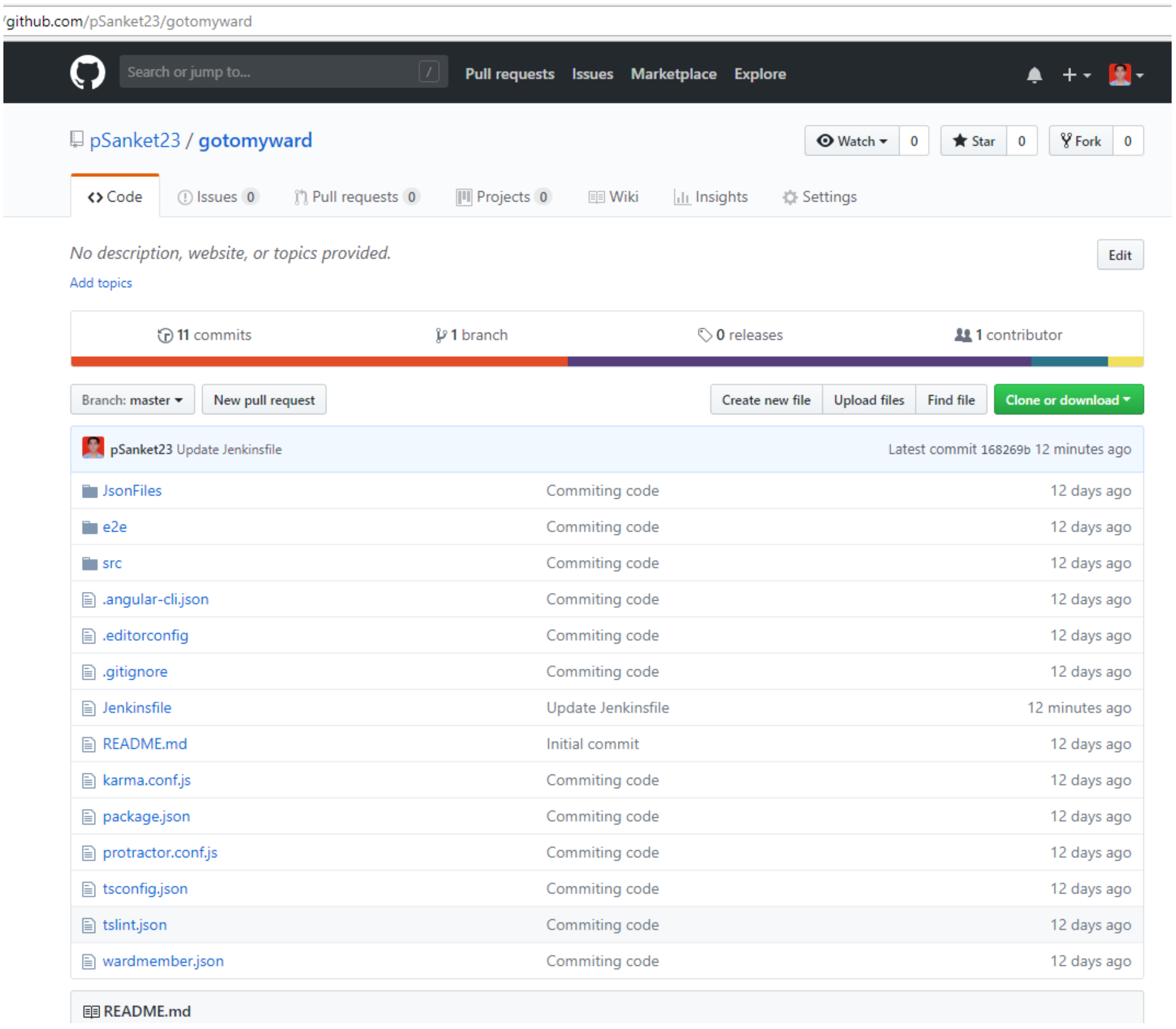


Console Output

```
Started by user Sanket Suresh Pethkar
Obtained Jenkinsfile from git git@github.com:pSanket23/gotomyward.git
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/gotomyward
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Clear Workspace)
[Pipeline] echo
Clear Workspace
[Pipeline] deleteDir
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Checkout)
[Pipeline] echo
Cloning Git Repository
[Pipeline] checkout
Cloning the remote Git repository
Cloning repository git@github.com:pSanket23/gotomyward.git
 > git init /var/lib/jenkins/workspace/gotomyward # timeout=10
Fetching upstream changes from git@github.com:pSanket23/gotomyward.git
 > git --version # timeout=10
using GIT_SSH to set credentials
 > git fetch --tags --progress git@github.com:pSanket23/gotomyward.git +refs/heads/*:refs/remotes/origin/*
 > git config remote.origin.url git@github.com:pSanket23/gotomyward.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url git@github.com:pSanket23/gotomyward.git # timeout=10
Fetching upstream changes from git@github.com:pSanket23/gotomyward.git
using GIT_SSH to set credentials
 > git fetch --tags --progress git@github.com:pSanket23/gotomyward.git +refs/heads/*:refs/remotes/origin/*
```

9. Screenshots.

GitHub Repository.



github.com/pSanket23/gotomyward

Search or jump to... Pull requests Issues Marketplace Explore

pSanket23 / gotomyward

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Add topics

11 commits 1 branch 0 releases 1 contributor

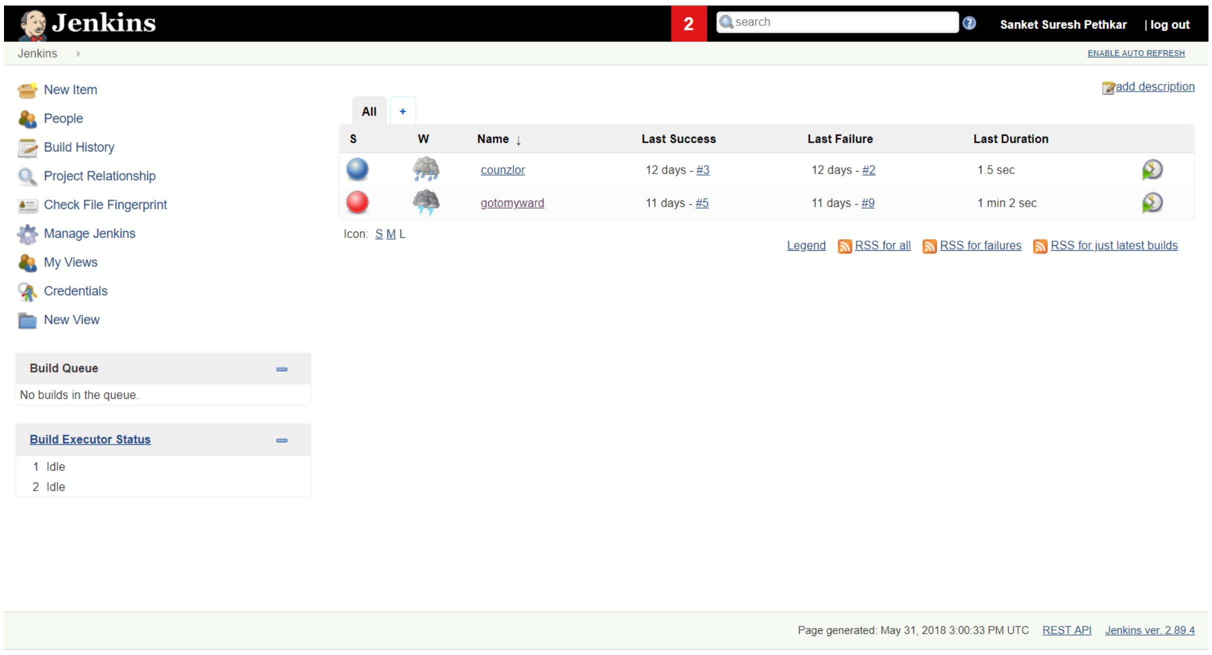
Branch: master New pull request Create new file Upload files Find file Clone or download

pSanket23 Update Jenkinsfile Latest commit 168269b 12 minutes ago

JsonFiles	Commiting code	12 days ago
e2e	Commiting code	12 days ago
src	Commiting code	12 days ago
.angular-cli.json	Commiting code	12 days ago
.editorconfig	Commiting code	12 days ago
.gitignore	Commiting code	12 days ago
Jenkinsfile	Update Jenkinsfile	12 minutes ago
README.md	Initial commit	12 days ago
karma.conf.js	Commiting code	12 days ago
package.json	Commiting code	12 days ago
protractor.conf.js	Commiting code	12 days ago
tsconfig.json	Commiting code	12 days ago
tslint.json	Commiting code	12 days ago
wardmember.json	Commiting code	12 days ago

README.md

Jenkins Homepage



Jenkins | 2 | search | Sanket Suresh Pethkar | log out

Jenkins › ENABLE AUTO REFRESH

- New Item
- People
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- My Views
- Credentials
- New View

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

add description

All | +

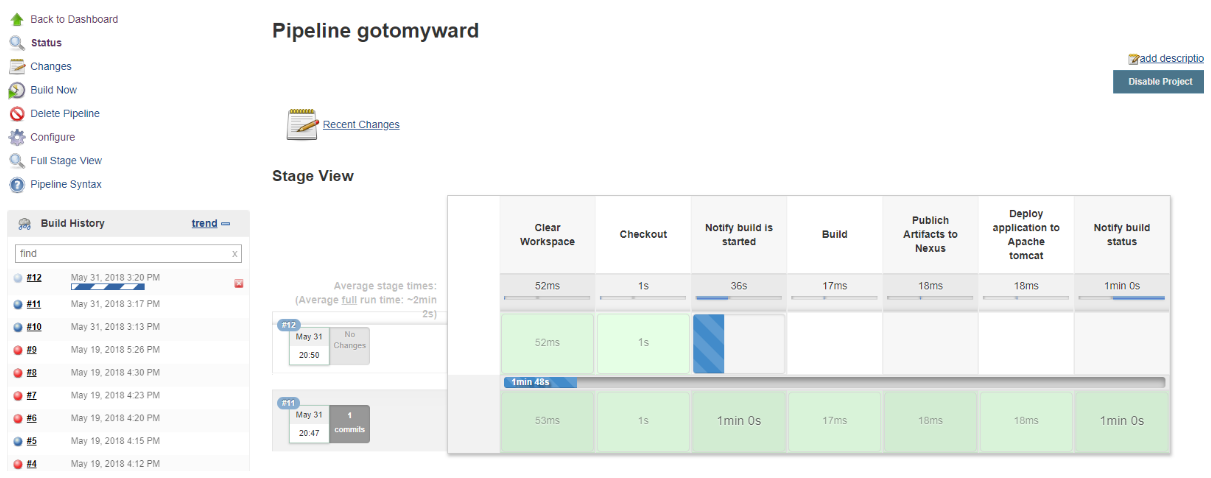
S	W	Name ↓	Last Success	Last Failure	Last Duration	
🔵		counzlor	12 days - #3	12 days - #2	1.5 sec	
🔴		gotomyward	11 days - #5	11 days - #9	1 min 2 sec	

Icon: S M L

Legend | RSS for all | RSS for failures | RSS for just latest builds

Page generated: May 31, 2018 3:00:33 PM UTC | REST API | Jenkins ver. 2.89.4

Jenkins Pipeline



- Back to Dashboard
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Full Stage View
- Pipeline Syntax

Build History trend

find

⚪	#12	May 31, 2018 3:20 PM
🔵	#11	May 31, 2018 3:17 PM
🔵	#10	May 31, 2018 3:13 PM
🔴	#9	May 19, 2018 5:26 PM
🔴	#8	May 19, 2018 4:30 PM
🔴	#7	May 19, 2018 4:23 PM
🔴	#6	May 19, 2018 4:20 PM
🔵	#5	May 19, 2018 4:15 PM
🔴	#4	May 19, 2018 4:12 PM

Pipeline gotomyward

add descriptio

Disable Project

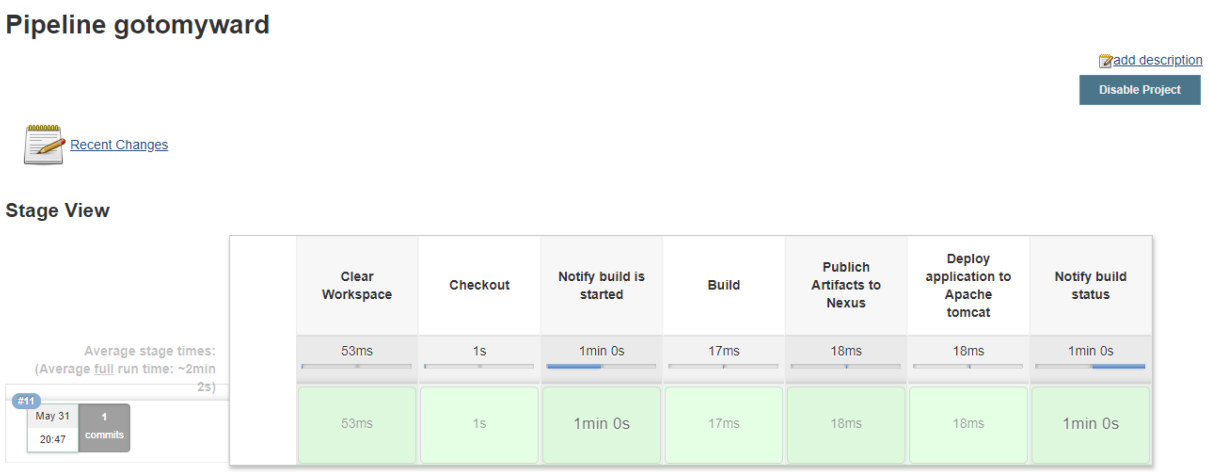
Recent Changes

Stage View

	Clear Workspace	Checkout	Notify build is started	Build	Publich Artifacts to Nexus	Deploy application to Apache tomcat	Notify build status
Average stage times: (Average full run time: ~2min 2s)	52ms	1s	36s	17ms	18ms	18ms	1min 0s
#12 May 31 20:50 No Changes	52ms	1s					
#11 May 31 20:47 1 commits	53ms	1s	1min 0s	17ms	18ms	18ms	1min 0s

1min 48s

Successful Job



Pipeline gotomyward

add description

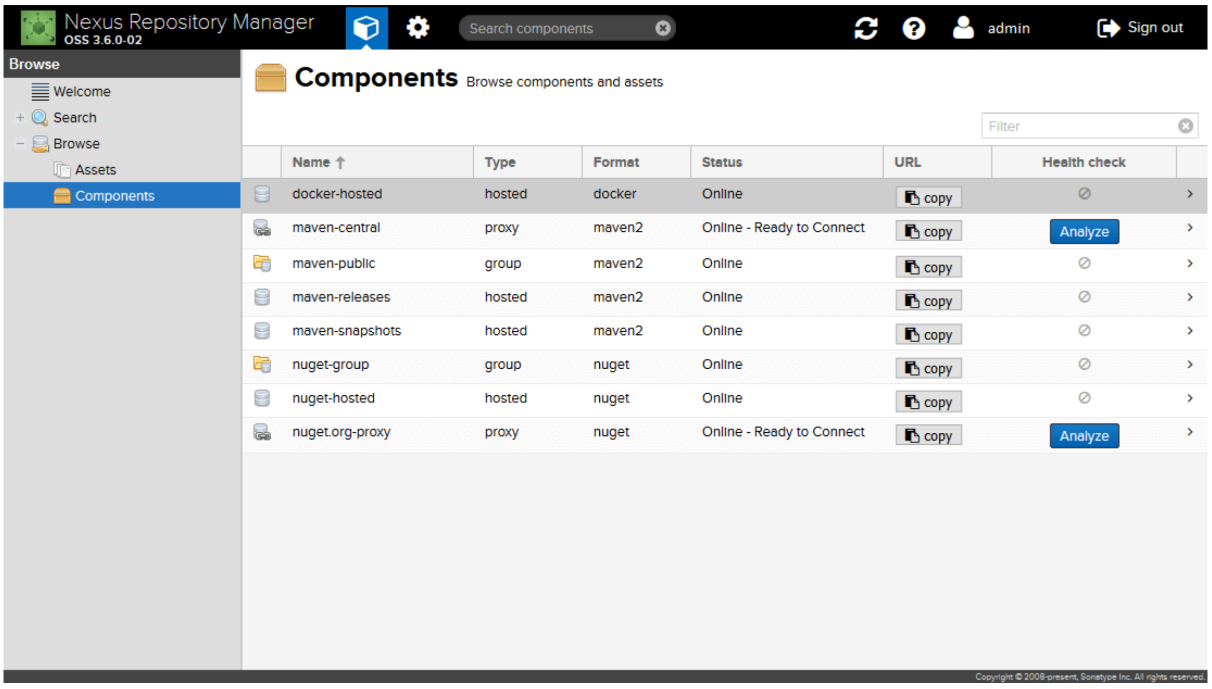
Disable Project

Recent Changes

Stage View

	Clear Workspace	Checkout	Notify build is started	Build	Publich Artifacts to Nexus	Deploy application to Apache tomcat	Notify build status
Average stage times: (Average full run time: ~2min 2s)	53ms	1s	1min 0s	17ms	18ms	18ms	1min 0s
#11 May 31 20:47 1 commits	53ms	1s	1min 0s	17ms	18ms	18ms	1min 0s

Nexus



Nexus Repository Manager OSS 3.6.0-02

Search components

admin Sign out

Browse
- Welcome
- Search
- Browse
  - Assets
  - Components

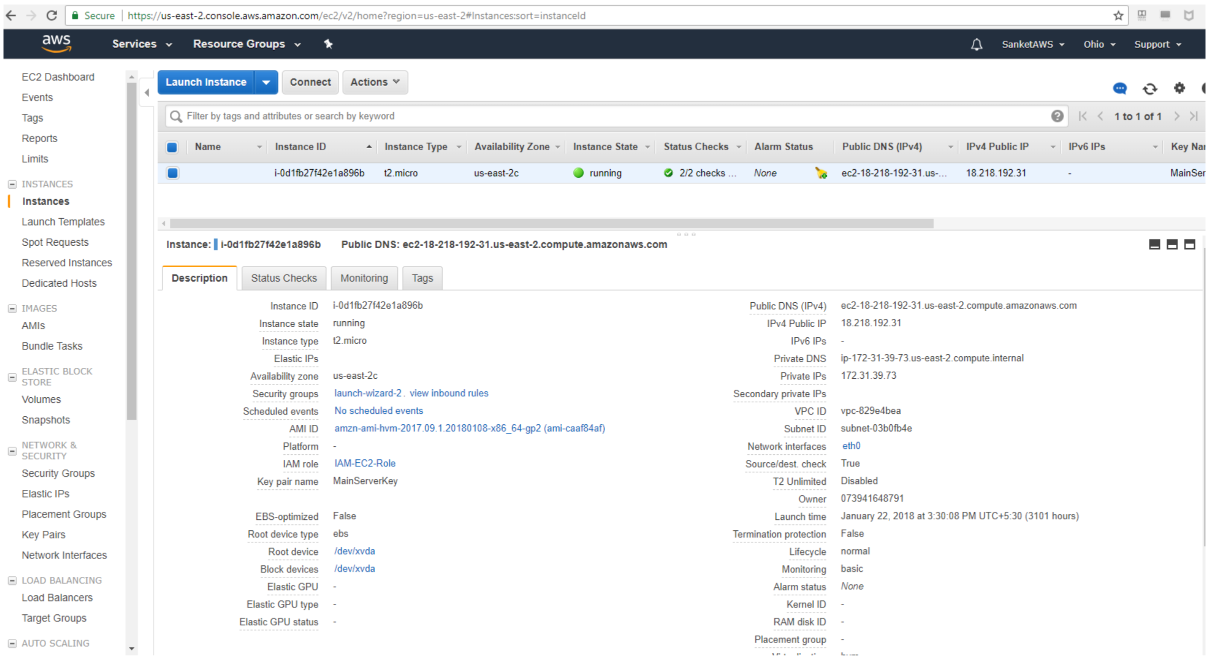
Components Browse components and assets

Filter

Name	Type	Format	Status	URL	Health check
docker-hosted	hosted	docker	Online	copy	⊘
maven-central	proxy	maven2	Online - Ready to Connect	copy	Analyze
maven-public	group	maven2	Online	copy	⊘
maven-releases	hosted	maven2	Online	copy	⊘
maven-snapshots	hosted	maven2	Online	copy	⊘
nuget-group	group	nuget	Online	copy	⊘
nuget-hosted	hosted	nuget	Online	copy	⊘
nuget.org-proxy	proxy	nuget	Online - Ready to Connect	copy	Analyze

Copyright © 2008-present, Sonatype Inc. All rights reserved.

AWS EC2 instance



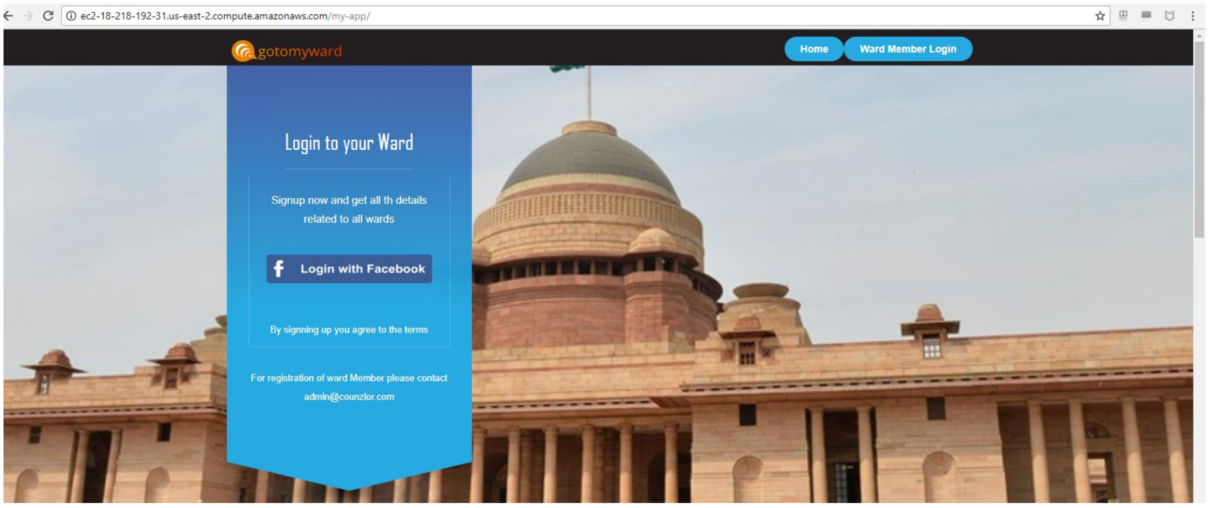
https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances:sort=instanceId

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Nar
	i-0d1fb27f42e1a896b	t2.micro	us-east-2c	running	2/2 checks ...	None	ec2-18-218-192-31.us-...	18.218.192.31	-	MainSer

Instance: i-0d1fb27f42e1a896b **Public DNS: ec2-18-218-192-31.us-east-2.compute.amazonaws.com**

Instance ID	i-0d1fb27f42e1a896b	Public DNS (IPv4)	ec2-18-218-192-31.us-east-2.compute.amazonaws.com
Instance state	running	IPv4 Public IP	18.218.192.31
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-39-73.us-east-2.compute.internal
Availability zone	us-east-2c	Private IPs	172.31.39.73
Security groups	launch-wizard-2 . view inbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-829e4bea
AMI ID	amzn-ami-hvm-2017.09.1.20180108-x86_64-gp2 (ami-caaf84af)	Subnet ID	subnet-03b0fb4e
Platform	-	Network interfaces	eth0
IAM role	IAM-EC2-Role	Source/dest. check	True
Key pair name	MainServerKey	T2 Unlimited	Disabled
		Owner	073941648791
EBS-optimized	False	Launch time	January 22, 2018 at 3:30:08 PM UTC+5:30 (3101 hours)
Root device type	ebs	Termination protection	False
Root device	/dev/xvda	Lifecycle	normal
Block devices	/dev/xvda	Monitoring	basic
Elastic GPU	-	Alarm status	None
Elastic GPU type	-	Kernel ID	-
Elastic GPU status	-	RAM disk ID	-
		Placement group	-

Application Deployed to AWS EC2 server



ec2-18-218-192-31.us-east-2.compute.amazonaws.com/my-app/

Login to your Ward

Signup now and get all th details related to all wards

By signning up you agree to the terms

For registration of ward Member please contact admin@counzlor.com

10. Abbreviations

CI – Continuous Integration

CD – Continuous Delivery

HA – High Availability.

YML – Yet another mark-up language.

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/