08

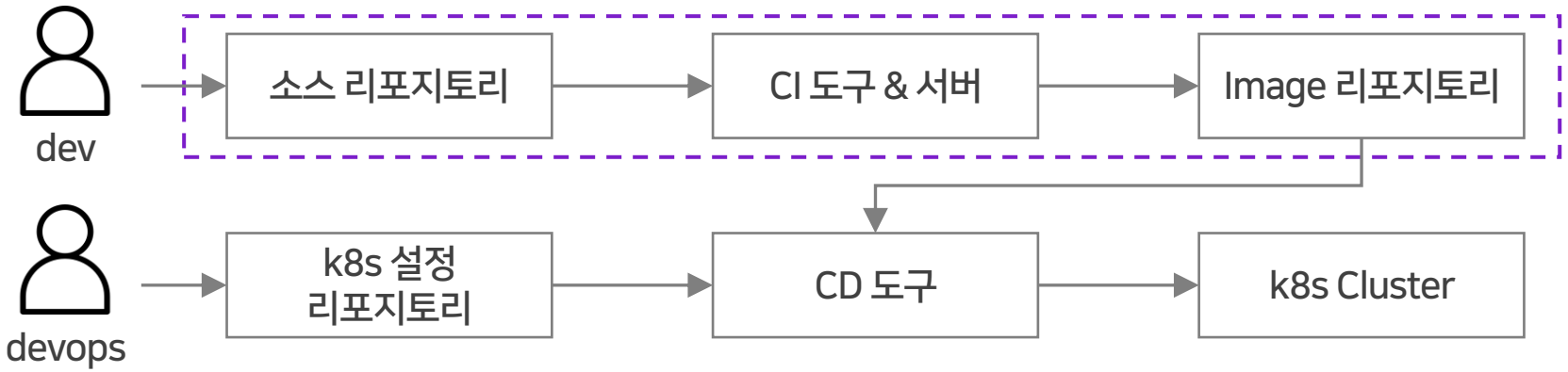
CI 자동화 도구

1. CI와 빌드 자동화
2. CI 계층별 사용 서비스
3. Jenkins 설치 및 구성
4. Item 프로젝트 생성-실행
5. Item 프로젝트 설정 상세
6. Jenkins To ECR
7. Jenkins 파이프라인 문법
8. Jenkins Agent 설정

1. CI와 빌드 자동화

I CI/CD on k8s

- CI(지속적 통합)
  - main 브랜치(또는 release 브랜치)에 새로운 버전의 코드가 병합되면...(Trigger!!)
  - CI 도구가 코드를 Pull 하여 Build --> Test를 수행함
  - 이렇게 Build 된 아티팩트(Docker Image)는 Image Repository(docker hub, ECR) 등에 Push



2. CI 계층별 사용 서비스

I CI 도구

AWS CodePipeline, Jenkins, Gitlab, Spinnaker

	Source Code Repository	Build	Test	Image Repository
Public or 자체	Github, Bitbucket	Jenkins, TeamCity, Codefresh		Docker Hub, Nexus
AWS	CodeCommit	CodeBuild		ECR
GCP	Cloud Source Repositories	Cloud Build		Container Registry
Azure	Azure Repos	Azure Pipelines		Azure Container Registry

3. Jenkins 설치 및 구성

l 설치 방법

- EC2 인스턴스에 리눅스 패키지를 이용해 설치
- SessionManager 를 이용해 터미널 환경에서 설치 시작

l 보안 그룹 검토

- jenkins-sg : 8080 포트 허용

l IAM Role 검토

- SessionManagerForEC2 : AmazonEC2RoleforSSM 관리형 정책 연결

3.1 Jenkins 설치

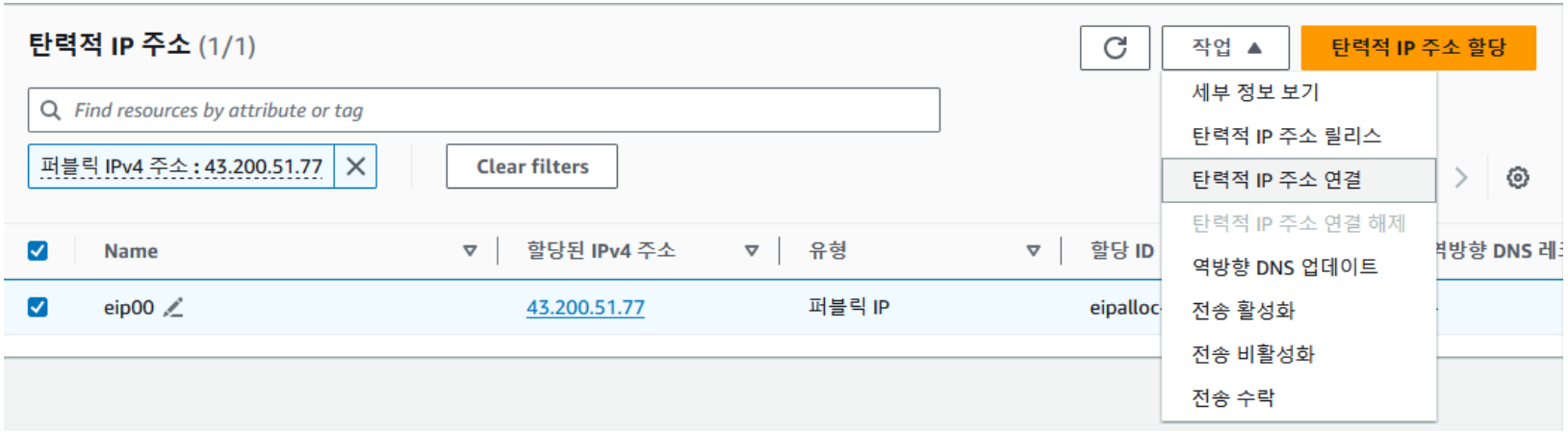
I EC2 인스턴스 생성

- 인스턴스명 : jenkinsNN
- AMI Image : Amazon Linux 2023 AMI
- 인스턴스 타입 : t3.small
- 키페어 : 키페어 없이 진행
- 보안그룹 : jenkins-sg
- 스토리지 구성 : 8Gib --> 20GiB로 변경
- 고급세부정보
  - IAM 인스턴스 프로파일 : SessionManagerForEC2
  - '인스턴스 시작'

3.1 Jenkins 설치

I 탄력적 IP 구성

- EC2 서비스로 이동 후 왼쪽 메뉴에서 '네트워크 및 보안 - 탄력적IP' 클릭
- '탄력적IP 주소 할당' 버튼 클릭
- '새로운 태그 추가' 버튼 클릭
  - Name : eipNN
  - eipNN 선택 후 '작업-탄력적 IP 주소 연결' 선택

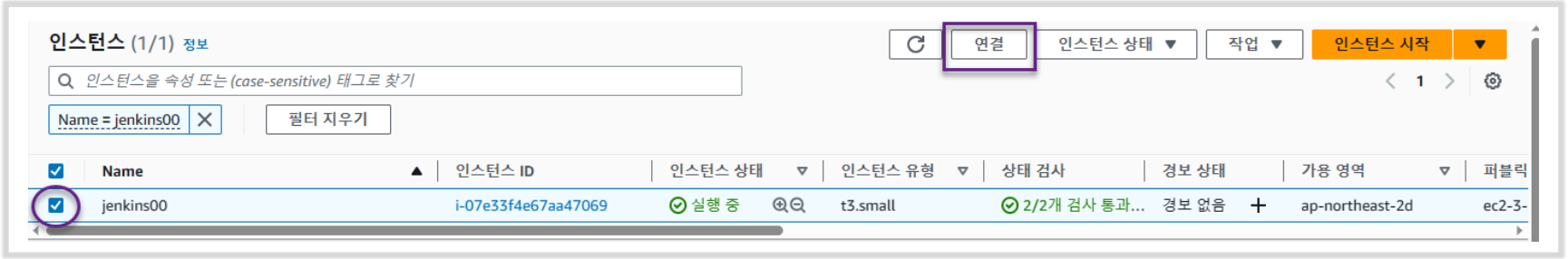


- 미리 생성한 jenkinsNN 인스턴스 선택 후 '연결' 버튼 클릭
- 탄력적 IP로 할당받은 IP주소가 Jenkins 서버의 고정 IP 주소가 됨

3.1 Jenkins 설치

I jenkinsNN 인스턴스에 접속

- EC2 서비스의 인스턴스 화면에서 jenkinsNN 인스턴스 체크박스 체크 후 '연결' 버튼 클릭



- Session Manager 탭에서 '연결' 버튼 클릭

3.1 Jenkins 설치

다음 명령어 실행하여 jenkins와 JDK 설치

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum update -y

sudo yum install java-17-amazon-corretto-devel -y
sudo yum install jenkins maven -y
sudo systemctl daemon-reload
```

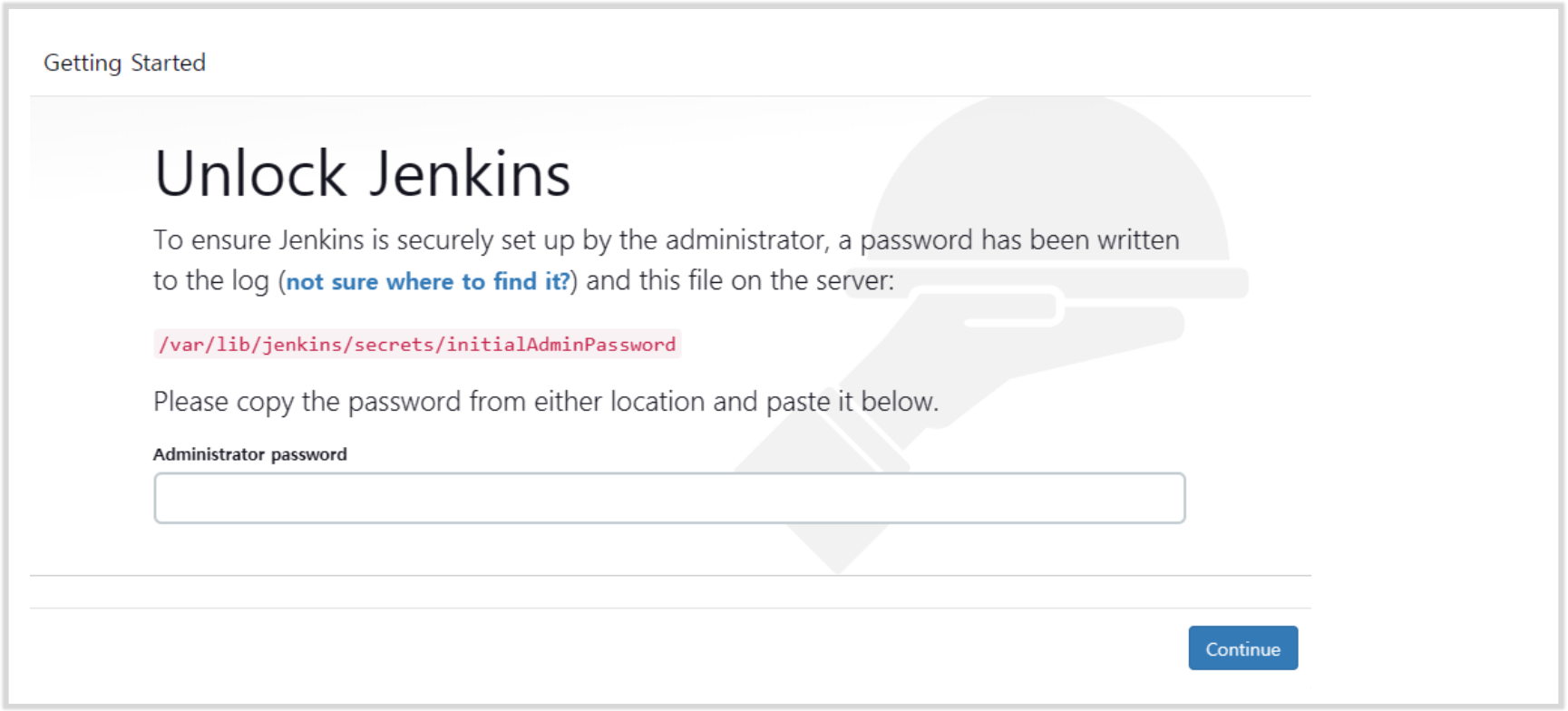
jenkins 서비스 실행, docker, git 설치

```
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo yum install git docker -y
sudo usermod -a -G docker jenkins
sudo systemctl enable docker
sudo service docker start
sudo chmod 666 /var/run/docker.sock
```

3.1 Jenkins 설치

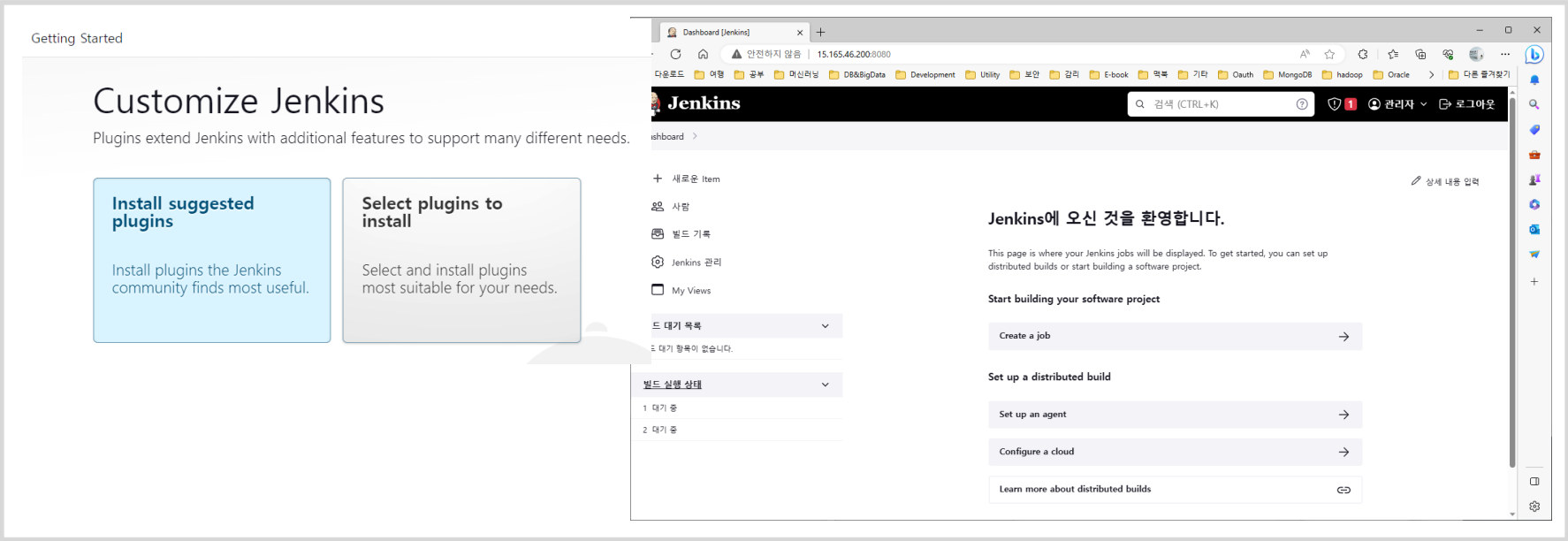
Ⅰ 브라우저로 Jenkins 접속하고 초기 패스워드 입력

- http://퍼블릭IP주소:8080
- 초기화면상의 경로에서 초기 패스워드 획득하여 입력
  - sudo cat /var/lib/jenkins/secrets/initialAdminPassword



3.1 Jenkins 설치

I 권장 플러그인 설치 진행



I Create First Admin User

- 계정명 : admin
- 암호 : adminpwd
- 이메일주소 : 자신의 이메일 주소

3.2 Jenkins 설정

Ⅰ 추가 플러그인 설치

- Jenkins Dashboard 화면에서 'Jenkins 관리' 링크 클릭
- System Configuration - Plugins 클릭
- Available plugins을 클릭하고 다음 플러그인을 검색하여 설치
  - Docker Pipeline
  - Docker Commons
  - Pipeline: Stage View
  - Build Timestamp
  - Amazon ECR
  - AWS Credentials
  - Nodejs

3.3 젠킨스 소스 github webhook 지정

github 계정과 Access Token 확인

- 만일 준비되지 않았다면 7장 9페이지 확인하여 계정 생성-> Access Token 생성
- 생성한 Access Token과 사용자 명을 잘 보관

github 리포지토리 생성

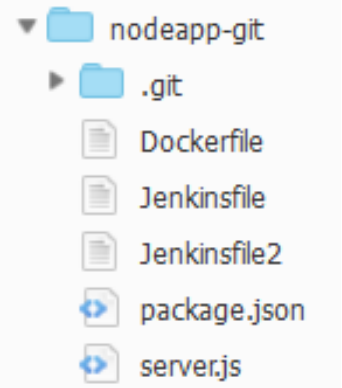
- nodeapp-git : private, 나머지 옵션을 private으로 설정

자신의 Cloud9 인스턴스에서 git clone

- UserName와 Password는 보관해둔 사용자명과 Access Token 입력

미리 제공된 nodeapp-git 예제 파일을 복사후 commit/push

- 명령어
  - git add .
  - git commit -am "초기화"
  - git push
- Password를 요구하는 경우 Access Token 입력
- 자격증명 캐싱하는 방법
  - git config --global credential.helper store
  - ~/.git-credentials 에 자격증명 저장됨.



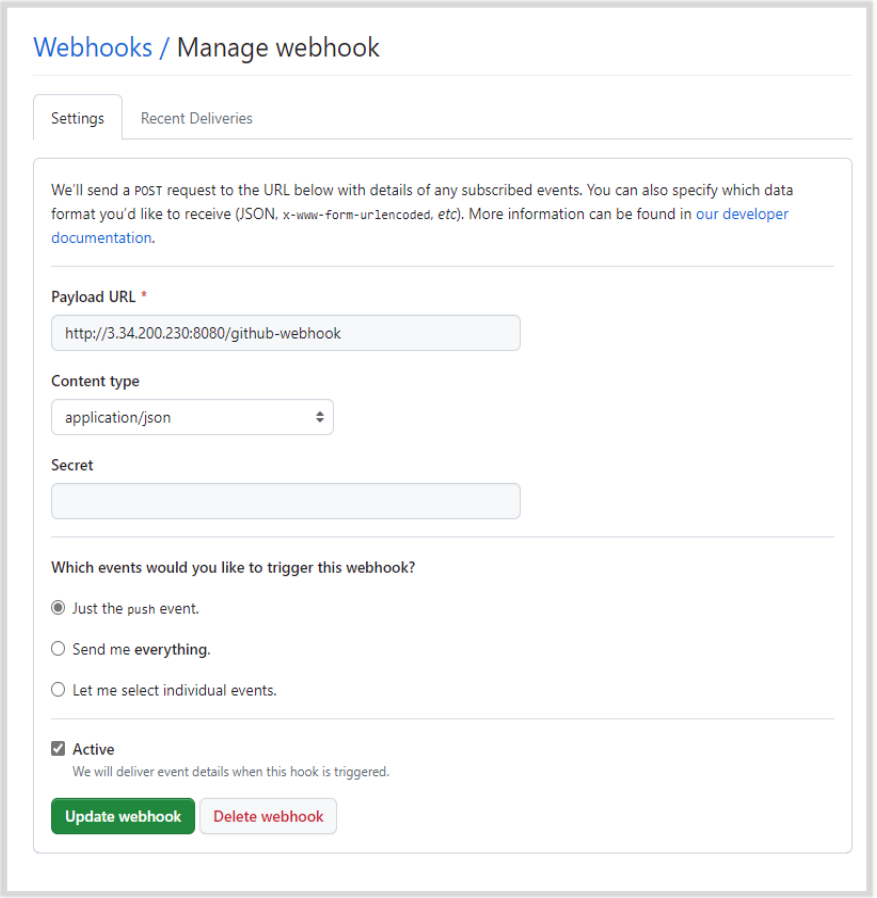
3.3 젠킨스 소스 github webhook 지정

I webhook 지정

- 생성한 nodeapp-git 리포지토리의 Settings 화면으로 이동
- 좌측 메뉴에서 webhooks 클릭
- 화면 상단의 Add webhook 버튼 클릭
- Payload URL에 다음 정보 입력 후 webhook 등록
  - http://JENKINS-PUBLIC_IP:8080 /github-webhook/

    마지막 슬래시 기호 반드시 입력

  - Content-type : application/json
  - 나머지는 기본값으로...



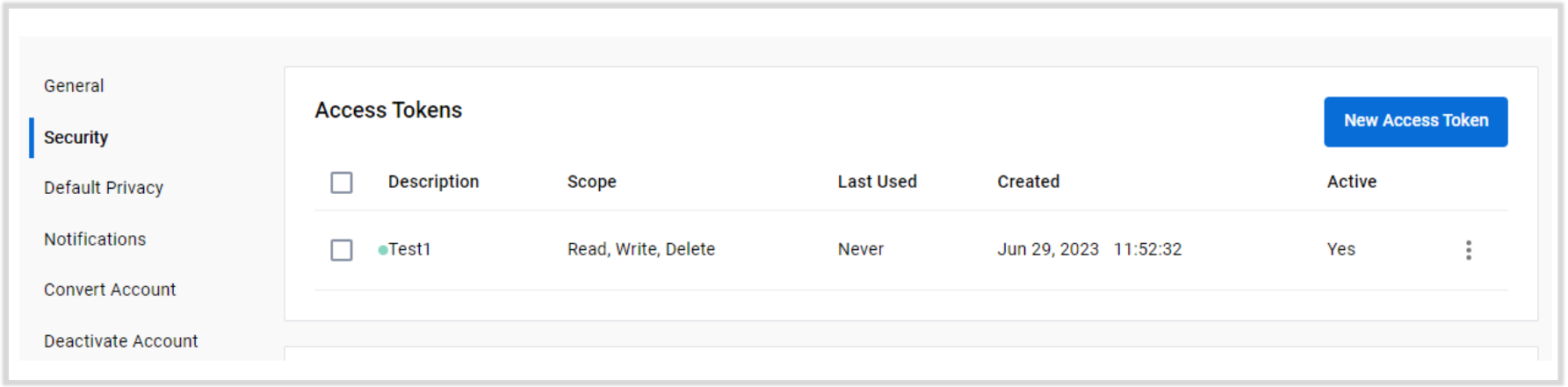
3.4 docker hub 계정 생성

I 6장에서 계정 생성 & Access Token 생성 완료

- 만일 생성하지 않았다면 6장 내용 참조해서 생성

I 생성 방법

- Docker 사용자 계정으로 로그인
- 우측 상단의 계정명 클릭 - Account Settings 클릭
- 왼쪽 메뉴에서 Security 클릭 후 New Access Token 클릭
- 적절한 권한 지정 후 생성하고 사용자명과 Access Token 잘 보관



3.5 Jenkins에 자격증명 등록 & Tool 설정

Jenkins 메인화면에서 Jenkins 관리 클릭

Security - Credentials 로 이동

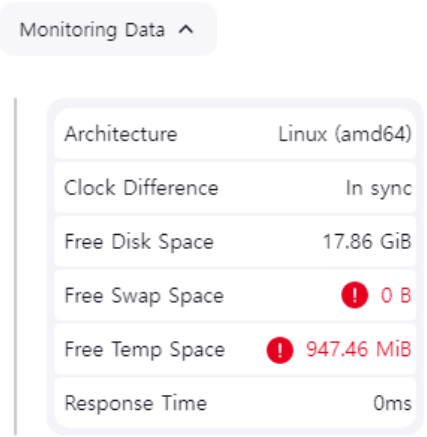
System Scope에서 Global Credentials로 이동하여 새로운 자격증명 등록

- docker hub credentials
  - 'Add Credentials' 버튼 클릭 후 New credentials 화면으로 전환
  - kind : Username with password 선택
  - Scope : Global
  - Username, Password : 미리 생성해둔 docker hub 자격증명 입력
  - ID : dockerhub-cred
  - Description: docker hub 자격증명
- github credentials
  - 'Add Credentials' 버튼 클릭 후 New credentials 화면으로 전환
  - kind : Username with password 선택
  - Scope : Global
  - Username, Password : 미리 생성해둔 github 자격증명 입력
  - ID : github-cred
  - Description: github 자격증명

3.6. Jenkins Node 설정

메인 화면에서 Jenkins 관리 - Nodes로 이동

- Built-in Node가 오프라인 상태임
- 원인
  - Free temp space(/tmp)가 부족
  - tmpfs는 임시 파일시스템으로 메모리의 일부를 사용
  - tmpfs의 사용가능한 공간이 jenkins가 정한 임계값 이하이기 때문
  - 근본적인 해결은 메모리를 늘려주어야 함
  - 이 과정에서는 임계값을 낮추어서 해결



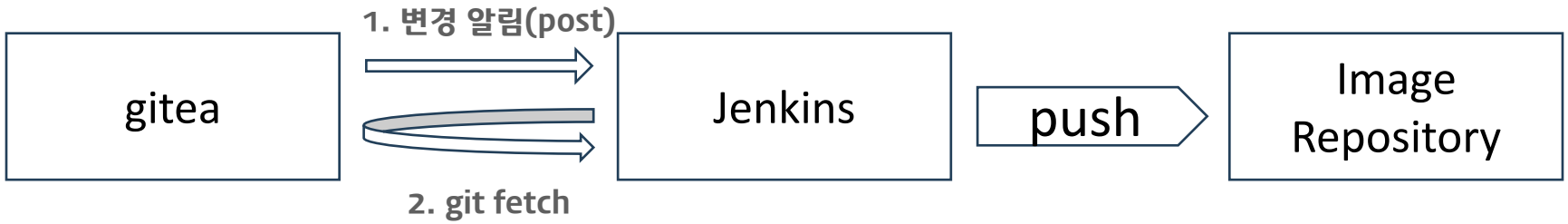
문제 해결

- Build-in Node의 설정으로 이동
- Node Properties에서 Disk Space Monitoring Threshold 항목을 체크
- Free Temp Space 의 Threshold 값을 500MB로 설정

4. Item 프로젝트 생성-실행

Trigger 방식 1 : Poll SCM

Trigger 방식 2 : Webhook



4. Item 프로젝트 생성-실행

I 대시 보드 화면에서 새로운 Item 클릭

- item name : nodeapp-ci-test
- Pipeline 선택 후 Ok 클릭
- General Section에서 github project 체크
  - '이 빌드는 매개변수가 있습니다' 체크 후 매개변수 추가

> String Parameter로 다음과 같이
> - Name: IMAGE_REPO
> - Default Value : [docker hub 사용자명]/nodeapp-git --> 예) gdhong/nodeapp-git

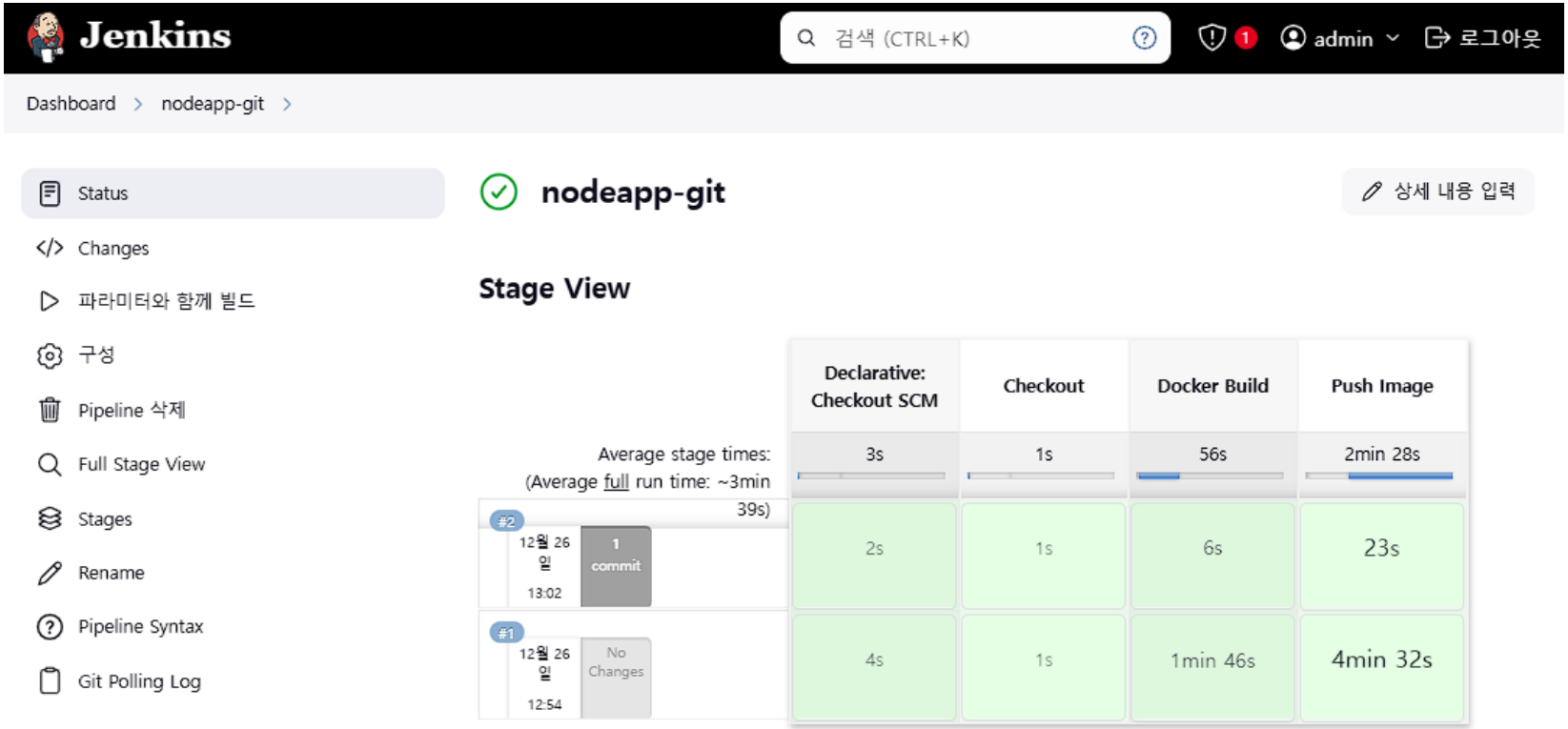
- Build Triggers 섹션에서 'github hook trigger for GITScm Polling' 선택
- Pipeline 섹션에서
  - Definition : Pipeline script from SCM, SCM : Git
  - Repository URL : https://github.com/[사용자명]/nodeapp-git
  - Credentials : github-cred 선택
  - Branch Specifier : */main

I 저장 후 완료

4. Item 프로젝트 생성-실행

I 실행

- '파라미터와 함께 빌드' 를 클릭하여 작동 여부 확인 후 webhook에 의한 자동 트리거 확인



5. Item 프로젝트 설정 상세

I General

- Do not allow concurrent builds
  - 동시 빌드를 허용하지 않으면 실행중인 빌드가 완료될 때까지 기다림.
  - 이전 빌드 중단 : 새로운 빌드가 시작되면 이전 빌드를 중단함
- Do not allow the pipeline to resume if the controller restarts
  - Jenkins가 재시작되었을 때 중단되었던 빌드 작업이 재개되지 않도록 함
- Throttle builds
  - 일정 시간 범위 내에 과도한 횟수의 빌드 작업이 일어나지 않도록 설정함
- 오래된 빌드 삭제
  - 전략 : Log Rotation
  - 빌드 이력 유지 기간 : 일자로 설정
  - 보관할 최대 갯수 : n개

5. Item 프로젝트 설정 상세

I Trigger

- Build after other projects are built
  - 다른 프로젝트의 빌드가 완료되면 새 빌드가 시작되도록 설정함
- Build periodically
  - 주기적으로 자동 빌드함. cron 대신에 사용하기 위한 것
- GitHub hook trigger for GITScm polling
  - Github push 훅을 이용해 빌드 작업을 트리거함. Github 플러그인이 지원함
- Poll SCM
  - SCM에서 변경 여부를 탐지하기 위해 주기적으로 폴링함. 성능에 나쁜 영향을 줄 수 있으므로 github hook 방식을 권장함
- Quiet period
  - 이 값을 0보다 큰 값을 설정하면 빌드 작업이 큐에 추가되지만 빌드 시작전에 지정된 시간(초)만큼 대기함.
  - 직접 빌드 버튼을 클릭하거나 원격 API를 이용해 호출하는 경우에만 적용됨.
  - 여러 사람이 동시에 같은 Job 을 실행시키는 경우 일정 시간을 대기하여 모아서 실행시킴
  - 기본값은 5초이며, 기본값을 변경하려면 "Jenkins 관리 / 시스템"에서 Quiet Period를 설정하면 됨.
- Remote API
  - 빌드 작업을 REST 방식으로 원격으로 트리거함. CSRF 공격의 위험때문에 권장하지 않음

5. Item 프로젝트 설정 상세

I Pipeline

- Pipeline script
  - 파이프라인 스크립트를 Item 프로젝트 내에서 직접 작성
- Pipeline script from SCM
  - 소스 제어도구로부터 파이프라인 스크립트(예: jenkinsfile)를 가져옴
  - 브랜치와 Script 경로를 지정함
- Light Checkout
  - 이것을 체크하면 전체 체크아웃을 수행하지 않고 직접 Jenkinsfile 스크립트만 가져오려고 시도함
  - 리포지토리의 아이템이 많을 때 효과적임

6. Jenkins To ECR

Ⅰ 필요 플러그인(이미 설치)

- Amazon ECR 플러그인
- AWS Credentials 플러그인

Ⅰ 자격증명 추가

- IAM 서비스로 이동하여 사용자에 대한 AccessKeyID + Secret AccessKey를 생성
- Jenkins 메인화면에서 Jenkins 관리 클릭 후 Security - Credentials 로 이동
  - 'Add Credentials' 버튼 클릭 후 New credentials 화면으로 전환
  - kind : AWS Credentials 선택
  - Scope : Global
  - Username, Password : 미리 생성해둔 docker hub 자격증명 입력
  - ID : aws-cred
  - Description : AWS 자격증명

6. Jenkins To ECR

I ECR에 접근할 때도 AWS IAM 접근 권한 필요

- nodeapp-git 리포지토리의 jenkinsfil2 참조
- Jenkins가 ECR에 접근하기 위해 AWS Credentials 이용
- ECR의 생성 후 레지스트리 + 리포지토리 등록
  - 예시) 111122223333.dkr.ecr.ap-northeast-2.amazonaws.com/nodeappNN

    레지스트리 리포지토리
- Jenkinsfile2 예제 참조

```
def IMAGE_VERSION = '1.0.0'
def ECR_REGISTRY = '111122223333.dkr.ecr.ap-northeast-2.amazonaws.com'
def ECR_REPO = "${ECR_REGISTRY}/nodeappNN"
......
app2 = docker.build("${ECR_REPO}")
......
docker.withRegistry("https://${ECR_REGISTRY}", 'ecr:ap-northeast-2:aws-cred') {
    app2.push(IMAGE_VERSION)
    app2.push("latest")
}
```

7. Jenkins 파이프라인 문법

I nodeapp-git 디렉토리의 Jenkinsfile 검토

```
def IMAGE_VERSION
pipeline {
  agent any
  stages {
    stage("Checkout") {
      steps {
        checkout scm
      }
    }
    stage('Docker Build') {
      agent any
      steps {
        script {
          IMAGE_VERSION = sh(script: "head -n 1 Dockerfile | sed 's/#//'",  returnStdout: true).trim()
          echo "${env.IMAGE_REPO}:${IMAGE_VERSION}"
          app = docker.build("${env.IMAGE_REPO}")
        }
      }
    }
    stage('Push Image') {
      agent any
      steps {
        script {
          docker.withRegistry("https://registry.hub.docker.com/${env.IMAGE_REPO}", "dockerhub-cred") {
            app.push(IMAGE_VERSION)
            app.push("latest")
          }
        }
      }
    }
  }
}
```

7. Jenkins 파이프라인 문법

Jenkins 파이프라인 작성 방법

- 스크립트 방법 : Scripted Pipeline
  - 범용 DSL인 Groovy 문법을 사용하여 빌드함
  - node와 stage라는 두개의 기본 블록만을 사용함

    node 블록 : 특정 파이프라인을 실행하는 머신 지정
    stage 블록 : 별도의 작업을 나타내는 단계를 그룹화함

- 선언적인 방법 : Declarative Pipeline
  - PAC : Pipeline-as-code
  - Scripted Pipeline보다 더 최신의 PAC 기법 제공
  - pipeline, agent, stages, stage 블록 사용
- 이 과정에서는 선언적인 방법 사용

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Build job is Completed'
      }
    }
    stage('Test') {
      steps {
         echo 'Test job is Completed'
      }
    }
  }
}
```

두 가지 유형

- section : 하나이상의 지시문(directive) 또는 step으로 구성
- directive : 지시문

```
node {
    stage('Build') {
        echo 'Build job is completed'
    }
    stage('Test') {
        echo 'Test job is completed'
    }
}
```

7. Jenkins 파이프라인 문법

section

- agent : master, slave 노드 중 어느 노드가 처리할지를 지정함.
- post : stage 실행이 완료될 때 실행할 단계 지정. 여러 조건 블록을 사용하여 작업 수행
- stages : 하나 이상의 stage 포함, 파이프라인의 작업을 정의함
- steps : stage에서 작업내의 세부 단계를 정의

7. Jenkins 파이프라인 문법

directive

- environment : pipeline, stage에서 사용할 k-v 형태의 환경 변수 정의
- options : pipeline, stage에서 사용할 다양한 옵션 지정
- parameters : pipeline 블록에서만 사용할 수 있는 파라미터 입력 받음
- triggers : pipeline 블록에서 자동화된 방식 지정, cron, pollSCM, upstream 방식 지정
- stage : pipeline에서 수행할 일들을 한 stage에 작성
- input : stage 내에서 사용자로부터 직접 입력을 받아야 할 때 사용
- tools : 빌드할 때 사용할 maven, gradle, jdk 지정
- when : stage가 실행될 조건을 정의, 하나 이상의 조건이 포함되어야 함.
  - when { branch pattern: "release-₩₩d+", comparator: "REGEXP"}
  - when { branch 'main' }

7. Jenkins 파이프라인 문법

I pipeline 블록

```
pipeline {
  /* 이 영역에 선언적 파이프라인 삽입 */
}
```

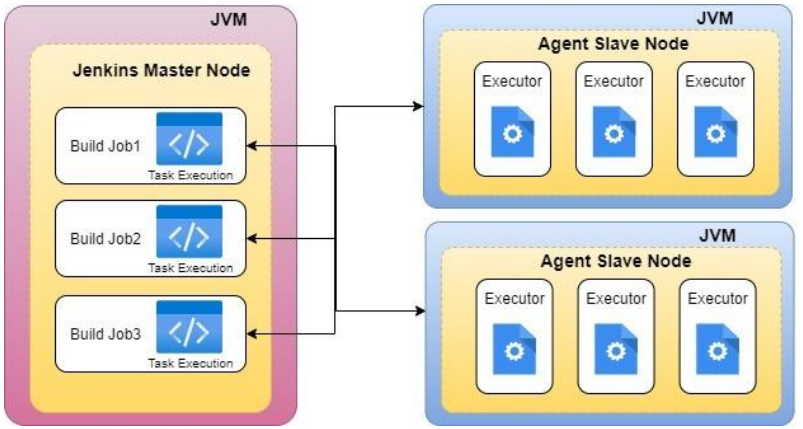
I agent directive

- 전체 pipeline 또는 특정 stage가 agent에서 작업을 진행할지를 지정
- jenkins에서 agent node 추가한 후 지정
  - Jenkins 분산 빌드를 위한 node 추가

    Master - Slave 구조

  - slave 노드에서는 java 11 이상이 설치되어야 함
  - 이 과정에서는 다루지 않음
  - 자세한 내용은 다음 문서 참조

    https://www.jenkins.io/doc/book/using/using-agents/



7. Jenkins 파이프라인 문법

- agent any
  - 사용가능한 agent에서 pipeline 또는 stage 실행
- agent none
  - top-level agent를 지정하지 않을 때 사용. 각 stage에서 agent를 지정함
- agent { label 'agent1' }
  - label로 지정한 agent에서 pipeline 또는 stage를 실행함

I SCM 관련 작업

- Jenkins SCM Step Plugin 활용 : 권장되는 플러그인으로 설치됨

```
pipeline {
    agent any
    stages {
        stage('checkout from SCM') {
            steps {
                checkout scm
            }
        }
    }
}
```

7. Jenkins 파이프라인 문법

Ⅰ Docker Image 관련 작업

- Docker Pipeline plugin 사용 권장
- k8s 앱 배포 시 해야 될 작업
  - 이미지 빌드
  - 이미지를 레지스트리로 푸시
- 이미지 빌드

```
//로컬 빌드
app = docker.build("nodeapp:${env.TAG}")
//원격 리포지토리로 푸시하기 위한 이미지 빌드
app = docker.build("111122223333.dkr.ecr.ap-northeast-2.amazonaws.com/nodeapp00:${env.TAG}")
```

- 이미지 푸시

```
docker.withRegistry("https://registry.hub.docker.com/stephenwon/nodeapp-git", "dockerhub-
credentials") {
   app.push(IMAGE_VERSION)
   app.push("latest")
}
docker.withRegistry("https://111122223333.dkr.ecr.ap-northeast-2.amazonaws.com/nodeapp",
'ecr:ap-northeast-2:user-AK') {
   app.push(IMAGE_VERSION)
   app.push("latest")
}
```

7. Jenkins 파이프라인 문법

I 환경 변수 적용 예

```
pipeline {
    agent any
    environment {
        IMAGE_VERSION = '1.0.1'
        IMAGE_REPO = 'stephenwon/nodeapp-git2'
    }
    stages {
        stage("###Checkout") {
            steps {
                checkout scm
            }
        }
        stage('###Docker Build') {
            agent any
            steps {
                script {
                    echo "${env.IMAGE_REPO}:${env.IMAGE_VERSION}"
                    app = docker.build("${env.IMAGE_REPO}")
                }
            }
        }
        stage('###Push Image') {
            steps {
                script {
                    docker.withRegistry("https://registry.hub.docker.com/${env.IMAGE_REPO}", "dockerhub-credentials") {
                        app.push(env.IMAGE_VERSION)
                        app.push("latest")
                    }
                }
            }
        }
    }
}
```

7. Jenkins 파이프라인 문법

I 수동 입력 요청

```
pipeline {
    agent any
    stages {
        stage('빌드 승인?') {
            input {
                message "빌드를 진행합니까? (Y/N)"
                ok "Build"
                submitter "PM, TL"
                parameters {
                    string(name: 'CONTINUE', defaultValue: 'Y', description: '빌드를 진행할지 Y/N으로 입력후 Build를 클릭합니다.')
                }
            }
            steps {
                echo "빌드 진행 : ${CONTINUE}"
            }
        }
    }
}
```

7. Jenkins 파이프라인 문법

I 일반적인 파이프라인 구조

- checkout
- 코드 정적 검사
  - 도구들

    SonarQube : 대표적인 코드 정적 검사 도구
    PMD
    Sparrow

  - SonarQube는 젠킨스 플러그인 지원

    https://waspro.tistory.com/596
    https://docs.sonarqube.org/latest/analyzing-source-code/scanners/jenkins-extension-sonarqube/

- 단위 테스트
  - JUnit Plugin --> build 파일 변경

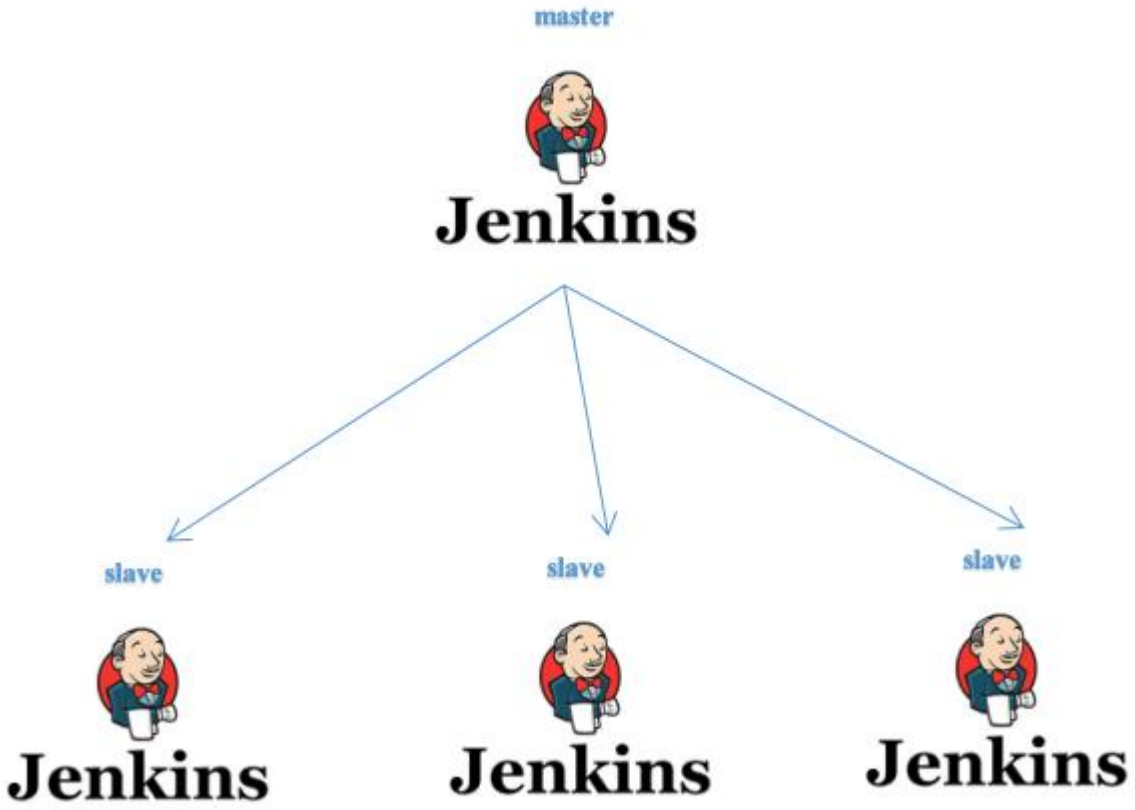
    단위 테스트 결과를 gigthub에 게시할 수 있음

  - Nodejs Plugin --> npm test
  - 단위 테스트는 image 빌드 단계에서 수행할 수도 있음

```
pipeline {
    agent any
    stages {
        stage('checkout SCM') {
        }

        stage('Static Check') {
        }

        stage('Unit Test') {
        }

        stage('Build') {
        }

        stage('Push') {
        }
    }
}
```

8. Jenkins Agent 설정

I Jenkins Agent 구성



8. Jenkins Agent 설정

Jenkins Agent 필수 조건

- JDK 11 이상이 반드시 설치되어 있어야 함

ssh private key 생성

- master에서 ssh-keygen -t rsa 명령 실행하여 Key pair 생성
- 생성된 키 페어중 Public Key를 ssh-copy-id slave1 명령으로 Slave 로 복사

노드간 시간이 일치하지 않을 경우를 위해 시간 서버를 위한 동기화 설정

- 예) sudo timedatectl set-ntp yes

8. Jenkins Agent 설정

Ⅰ Agent Node 추가

- master에서 ~/.ssh/id_rsa 파일 내부의 Private key를 복사하여 Jenkins의 'Jenkins관리 / Credentials' 에 ssh private key 로 등록
- 'Jenkins 관리 / Nodes' 로 이동하여 New Node 버튼 클릭
- 노드명 입력하고 Permanent Agent 지정 후 'create' 버튼 클릭
- Number of executors : 2
- Remote Root Directory : /home/사용자명/jenkins-agent (사용자 홈 디렉토리에 생성)
- Labels : agent 지정시 사용할 레이블 지정
- launch method : launch agent via SSH
- Host : slave 의 IP 주소 지정
- Credentials : Jenkins Credentials 에 등록한 자격증명 지정
- Host Key Verification Strategy : Manually trusted key verification strategy
- Availability : Keep this agent online as much as possible

8. Jenkins Agent 설정

I 설정된 Agent

Nodes

+ New Node | Configure Monitors

S	이름 ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	22.70 GiB	2.00 GiB	22.70 GiB	0ms
	slave1	Linux (amd64)	In sync	24.27 GiB	2.00 GiB	24.27 GiB	137ms
	slave2	Linux (amd64)	In sync	24.27 GiB	2.00 GiB	24.27 GiB	231ms
	수집된 데이터	**9 sec**	**8.9 sec**	**8.8 sec**	**8.7 sec**	**8.8 sec**	**8.8 sec**

9. Tools 설정

Tools 섹션

- agent none으로 설정된 경우에는 무시됨
- 지원되는 도구 : nodejs, maven, gradle, ant 등
- tools 설정과 이름은 'Jenkins 관리-Tools'에서 미리 설정되어 있어야 함
- 목적 : 도구의 자동 설치와 다중 버전 지원

```
pipeline {
    agent any
    tools {
        maven 'maven-3.6'
    }
    stages {
        stage('버전 확인') {
            steps {
                script {
                    sh 'mvn --version'
                }
            }
        }
    }
}
```

9. Tools 설정

Ⅰ Tools 기능 테스트

- Jenkins 관리 - tools 화면으로 이동
- NodeJS installations 로 이동하여 Add NodeJS 버튼 클릭
  - 만일 Add NodeJS가 보이지 않으면 NodeJS 플러그인을 추가해야 함
- 두개의 버전 추가 : name 속성이 중요함
  - name : node-24
    - install automatically 체크
    - NodeJS 24.x.x 버전 선택
  - name : node-22
    - install automatically 체크
    - NodeJS 22.x.x 버전 선택

9. Tools 설정

Ⅰ Item 프로젝트 생성

- 새로운 Item 프로젝트 생성
  - name : node-version-test
  - item type : Pipeline
- 구성
  - Pipeline 직접 지정

```
pipeline {
    agent any
    tools {
        nodejs 'node-24'
    }
    stages {
        stage('node 버전 확인') {
            steps {
                script {
                    sh 'node --version'
                }
            }
        }
    }
}
```

9. Tools 설정

I 실행 후 콘솔 로그 확인

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/node-version-test
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
Unpacking https://nodejs.org/dist/v24.4.1/node-v24.4.1-linux-x64.tar.gz to ......
[Pipeline] envVarsForTool
[Pipeline] }
.....(생략)
+ node --version
v24.4.1
.....(생략)
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
[Gitea] do not publish assets due to source being no GiteaSCMSource
Finished: SUCCESS
```

* 내용 정리

l k8s 환경에서 지속적 통합 도구

- 소스 리포지토리가 변경되면 Test를 거쳐 코드를 빌드하여 Docker Image를 생성하고 Image Repository로 Push 함

l 소스 리포지토리

- Github, Git, Bitbucket, Public Cloud가 제공하는 관리형 서비스 : 예) AWS CodeCommit

l CI 도구

- Jenkins, Public Cloud가 제공하는 관리형 서비스 : 예) AWS CodeBuild

l Image Repository

- Nexus, DockerHub, Public Cloud가 제공하는 관리형 서비스 : 예) AWS ECR