Jenkins

JOHN BRYCE
Leading in IT Education
matrix company

- MAKING A CHANGE -

●●●

Module 1 - JENKINS THEORY

Provided by DevopShift for JB & Presented by DevopShift CEO Yaniv Cohen

MY INFO

CV

https://goo.gl/9ivmRP

Linkedin

https://www.linkedin.com/in/yanivos/

AGENDA

- DEVOPS - INTRO
- Devops Hot Spots
- CI/CD

- INTRO -
THE PROBLEM

- Everything is software BUT it still runs on Servers to operate...
- Delivering a service is painfully slow
- Errors and faulty functionality
- Internal frictions between Product / R&D / IT creates the problem
- Delay in Delivery = Money
- The company culture is usually the bottleneck in the transition to “the DevOps concept”

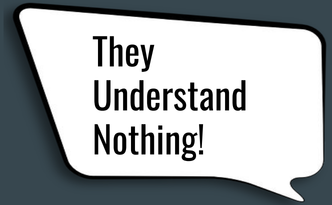
SYMPTOMS

- Faulty software services are release into production - Outages are expected
- No KPI and Monitoring facilitators are in place to help diagnose production issues Quickly and effectively.
- No Automated regression and load Tests
- No proper incident management and alerting in place From Alert to Postmortem.
- Blaming and shifting fingers pointing
- Long iteration from request to response for resources required by groups such QA , DEV to other teams and vice versa.
- “Manual errors / Human error” will be in most cases the ROOT CAUSE.
- Features are being released as “enabled” to the wild
- Life as an Ops engineer **sucks in your company**.

- INTRO -
WALL OF CONFUSION

Developers and IT Operations has a very different objective

Wall of confusion



Drives for **CHANGE**

DEVeloper

Drives for **STABILITY**

IT OPerator

- INTRO -
THE OBJECTIVE

COMPANY OBJECTIVE

IMPROVE TTM
- TIME TO MARKET -

TTM AGENDA

- Shorten the time from product to production delivery
- Cut down the time required for Technical and technological evolutions and implementations
- While keeping very high level of stability, quality, performance, cost and basicly 0 downtime…

COMPANY OBJECTIVE

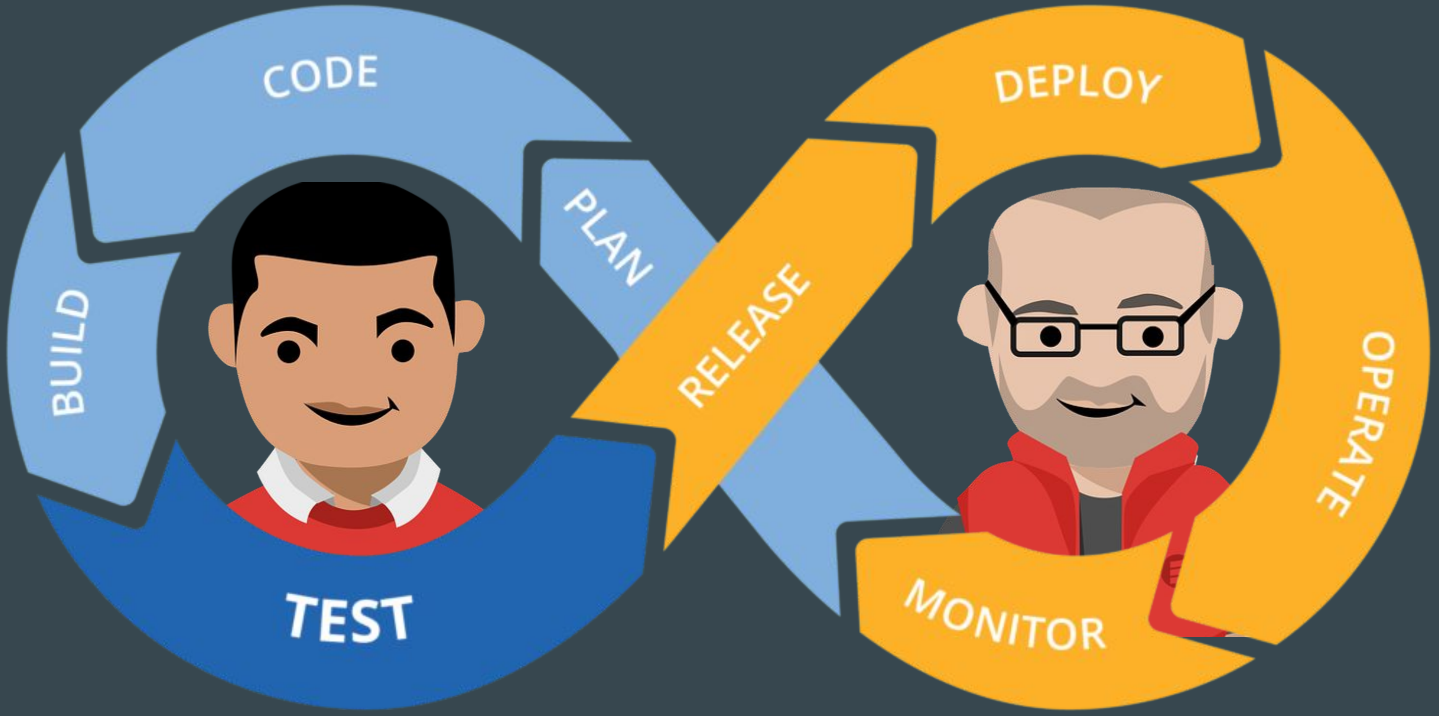
IMPROVE TTM
- TIME TO MARKET -

TTM PROBLEM

Two Worlds collided

- Improving TTM and improve quality are two opposing attributes in the development arena.
- Stability improvement is the **objective of IT Ops teams**
- Reducing/Improving TTM is the **objective of developers**

- INTRO -
SO HOW DO WE SOLVE IT?



Agile
Development

Agile
Deployment/Production

- INTRO -
SO WHAT IS DEVOPS?

DevOps was conceived from the idea of extending the Agile development practice.

Meaning:
Extending the software streamlining movement of Build , Validate and Deploy stages and tie it up with cross-functional teams from Design, Operation , Quality and Build automation, Delivery , Monitoring and thru production support.

DevOps (a clipped compound of "development" and "operations") is a software engineering practice that aims at unifying software development (Dev) and software operation (Ops).
*** source: WikipediA

- INTRO -
WHAT ARE THE EXPECTATION?

What do we expect to gain from a DevOps culture in our company and why...

Ideally

- Standardizing development environments
- IAC - Infrastructure as a code
- CD - Automate delivery process to improve delivery predictability , efficiency , security and maintainability
- Monitoring and Log shipping frameworks
- Culture of collaboration

Why should we ?

Provide **Developers** with more control of the production environment and a better understanding of the production infrastructure

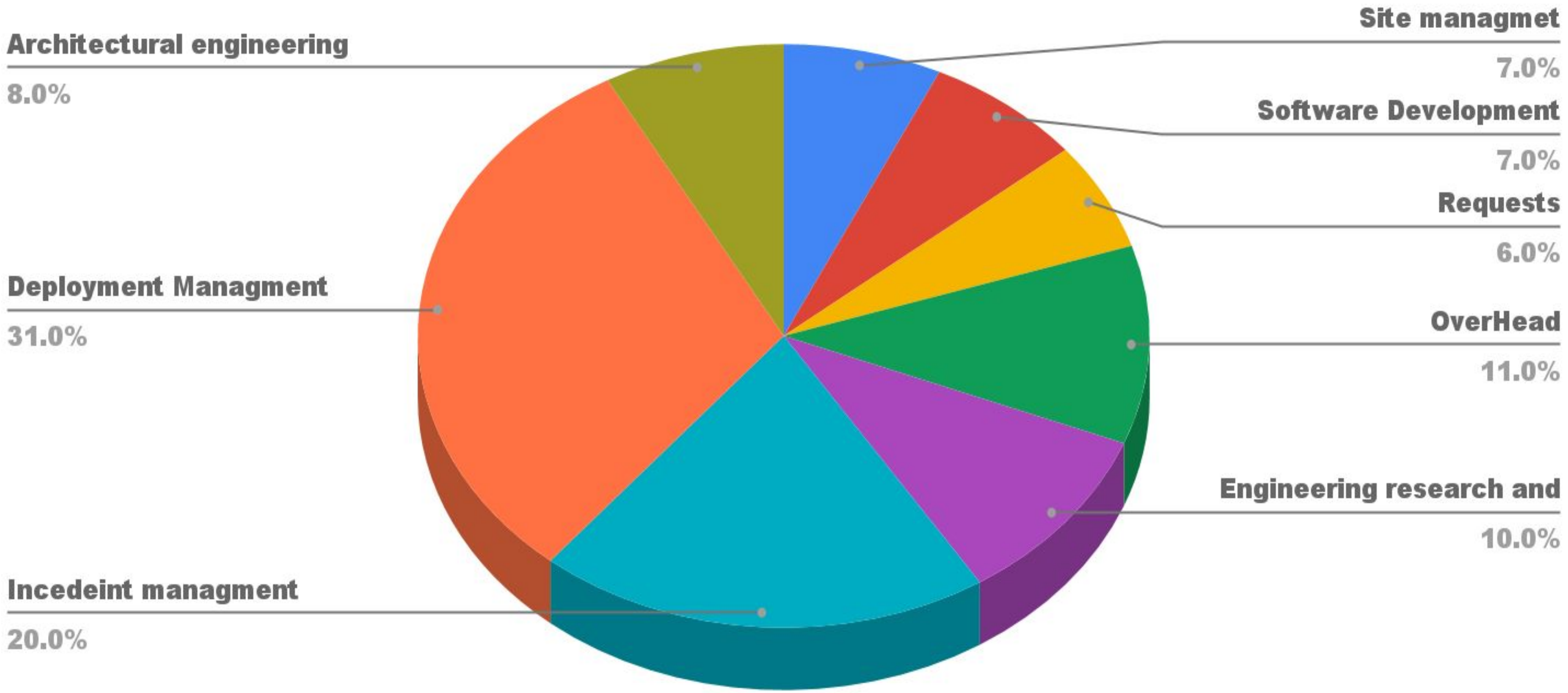
Interesting Facts - What does operation do?

Like most companies Operation invest almost 51% of their production time on **Deployments**

- Executing the deployment
- Resolving issues related to the deployments
- Post Mortem of outages occurred while and after deploying.

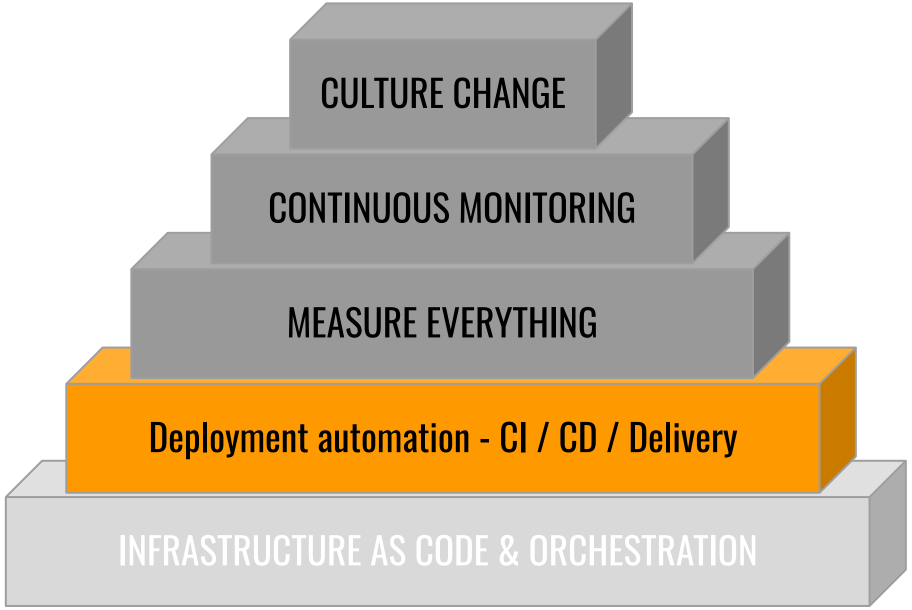
In a small startup built upon 3 - 5 IT Operation members the deployment cost can reach to a dazzling of **$140k - $210k** (out of $560k employees cost) year for only supporting deployments!

40% - 51% is for deployment & incident management



- MOVING FORWARD - DEVOPS HOTSPOTS

IMPLEMENTING DEVOPS FOUNDATIONS


CULTURE CHANGE
CONTINUOUS MONITORING
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS CODE & ORCHESTRATION

CONFIGURATION AS A CODE

- Automate everything
  - Infrastructure provisioning
  - Application deployment
  - Runtime Orchestration

- Build a development Workflow
  - Write it in code
  - Validate the code
  - Unit test the code
  - Build it into an artifact
  - Deploy artifact to test
  - Integration test it
  - Deploy artifact to prod

IAC AUTOMATION TOOLING

- IAC Models - AWS CloudFormation, **Terraform**, Azure ARM, Ubuntu Juju
- Hardware Provisioning - **Packer**, Foreman , MaaS, Crowbar...
- CM - Puppet , Chef , Ansible , Salt...
- Integration Testing - rspec, serverspec, Ansible integration tests
- Orchestration - Rundeck, Kubernetes (for Docker), **ECS**

- IAC -
TERRAFORM

Terraform is a radically simple IT automation engine that automates on premise and (multi) cloud provisioning, Infrastructure as a code management, *application deployment, intra-service orchestration, and many other IT needs.

Terraform allows users to define a datacenter infrastructure in a high-level configuration language, from which it can create an execution plan to build the infrastructure

- **Clear** - Simple declarative syntax written in json layout
- **Fast** - Learning curve is around 1 - 2 weeks for operating and 3 - 4 weeks to build top to bottom modules
- **Complete** - You really don't need anything else...
- **Efficient** - Extending functionality of Terraform is done using modules
- **Secure** - using IAM - nothing more than that is required

- IAC-
TERRAFORM

Terraform to control

- Production / STG / QA / DEVELOPMENT
  - VPC Creation and configuration
  - Network (such as ALB / SG / ASG)
  - Roles deployments Infrastructure & Code
  - Self Service INFRASTRUCTURE

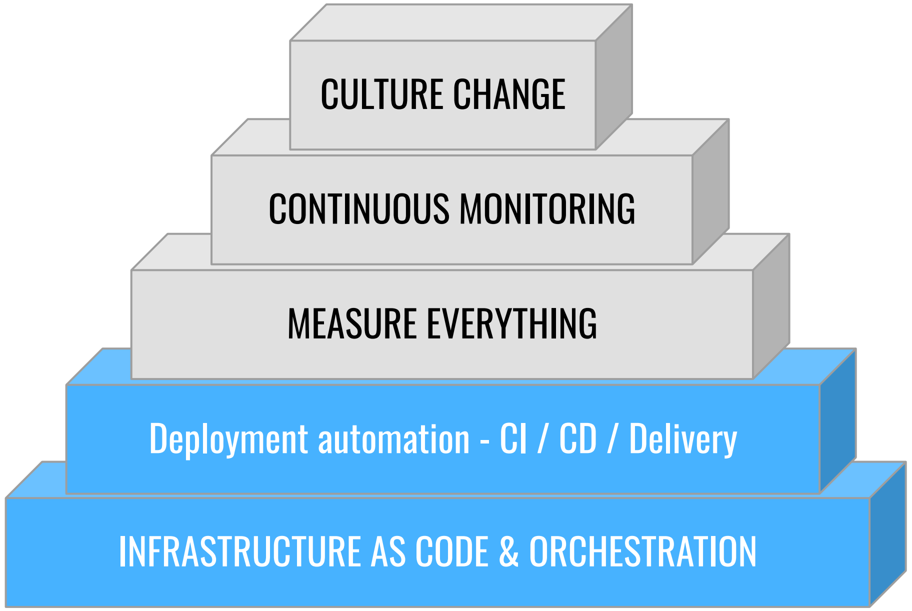
TERRAFORM SAVES

Hundreds of hours In Deploying , Maintaining , updating and controlling your infrastructure

Prior to terraform **Infrastructure** was unmanaged (as a code), Deployments was a pain in the ass, written with fabric / Ansible / Salt / Scripts

Which made it unscalable to adopt tech evolution such as K8S and containers

IMPLEMENTING DEVOPS FOUNDATIONS

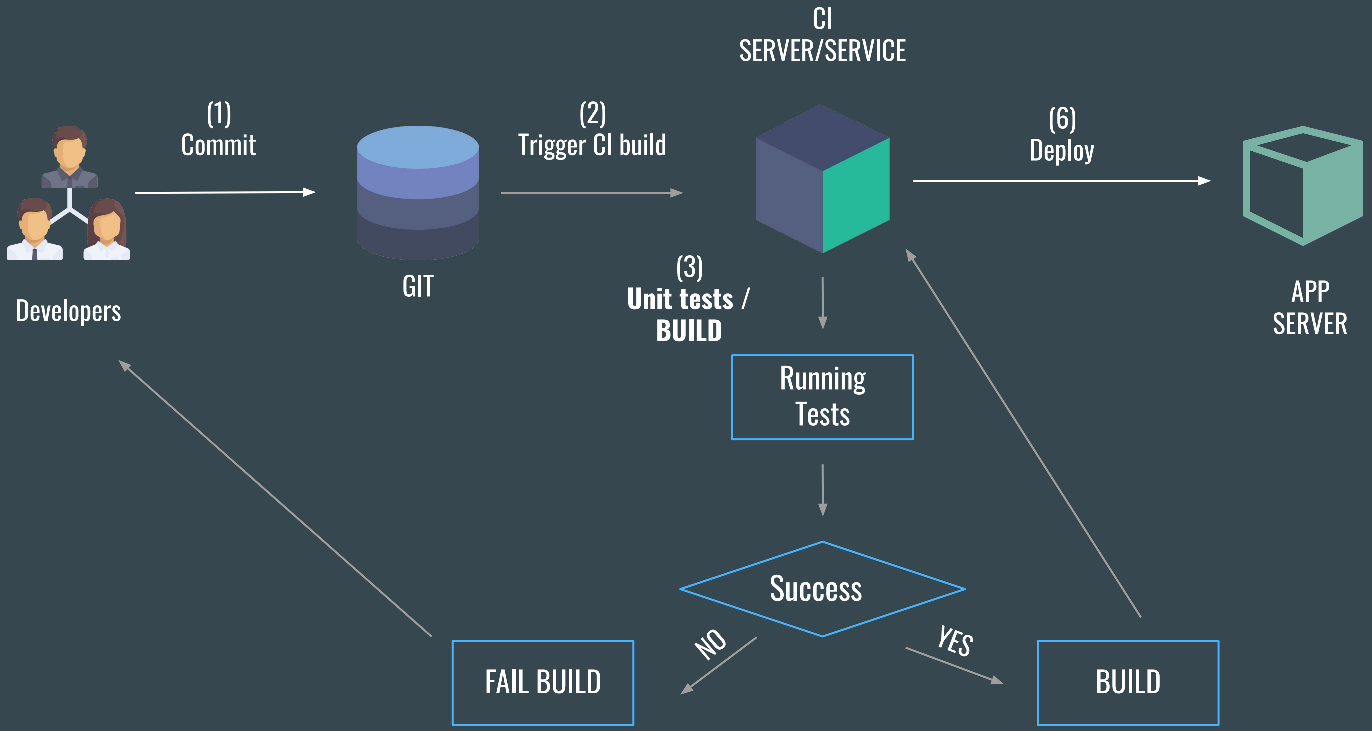

CULTURE CHANGE
CONTINUOUS MONITORING
MEASURE EVERYTHING
Deployment automation - CI / CD / Delivery
INFRASTRUCTURE AS CODE & ORCHESTRATION

CI - Continuous Integration

Continuous Integration (CI) is a software development practice that is based on a frequent integration of the code into a shared repository. Each check-in is then verified by an automated build.

main goal of continuous integration is to identify the problems that may occur during the development process earlier and more easily. If you integrate regularly—there is much less to check while looking for errors. That results in less time spent for debugging and more time for adding features.

BASIC CI FLOW

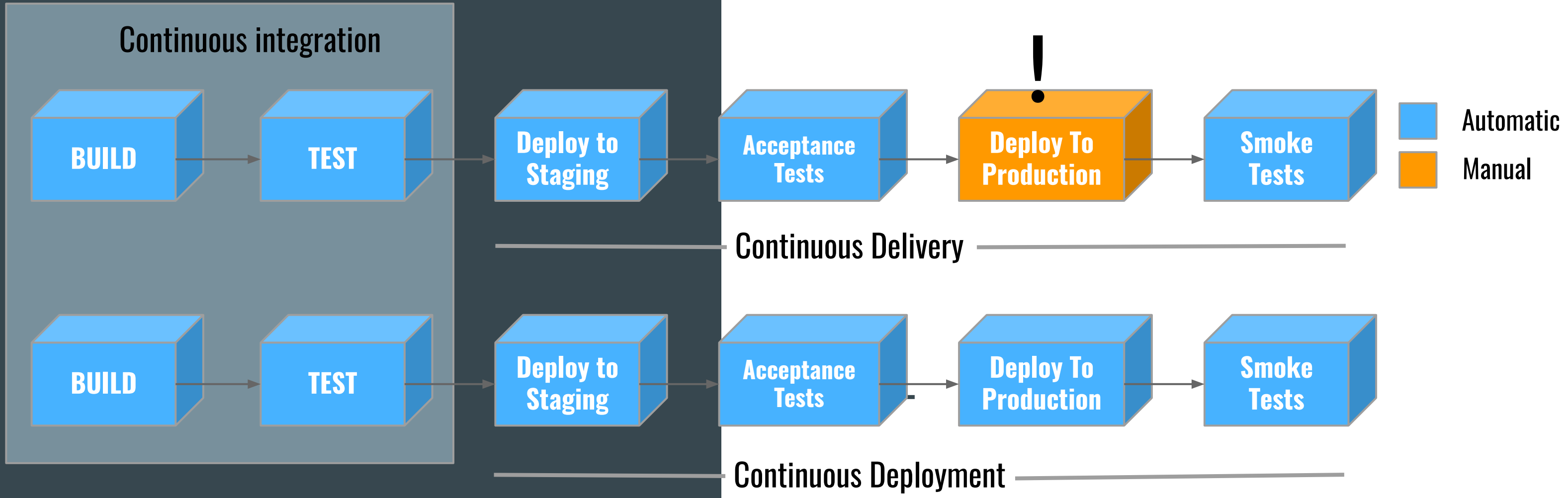


CONTINUOUS DELIVERY/DEPLOYMENT

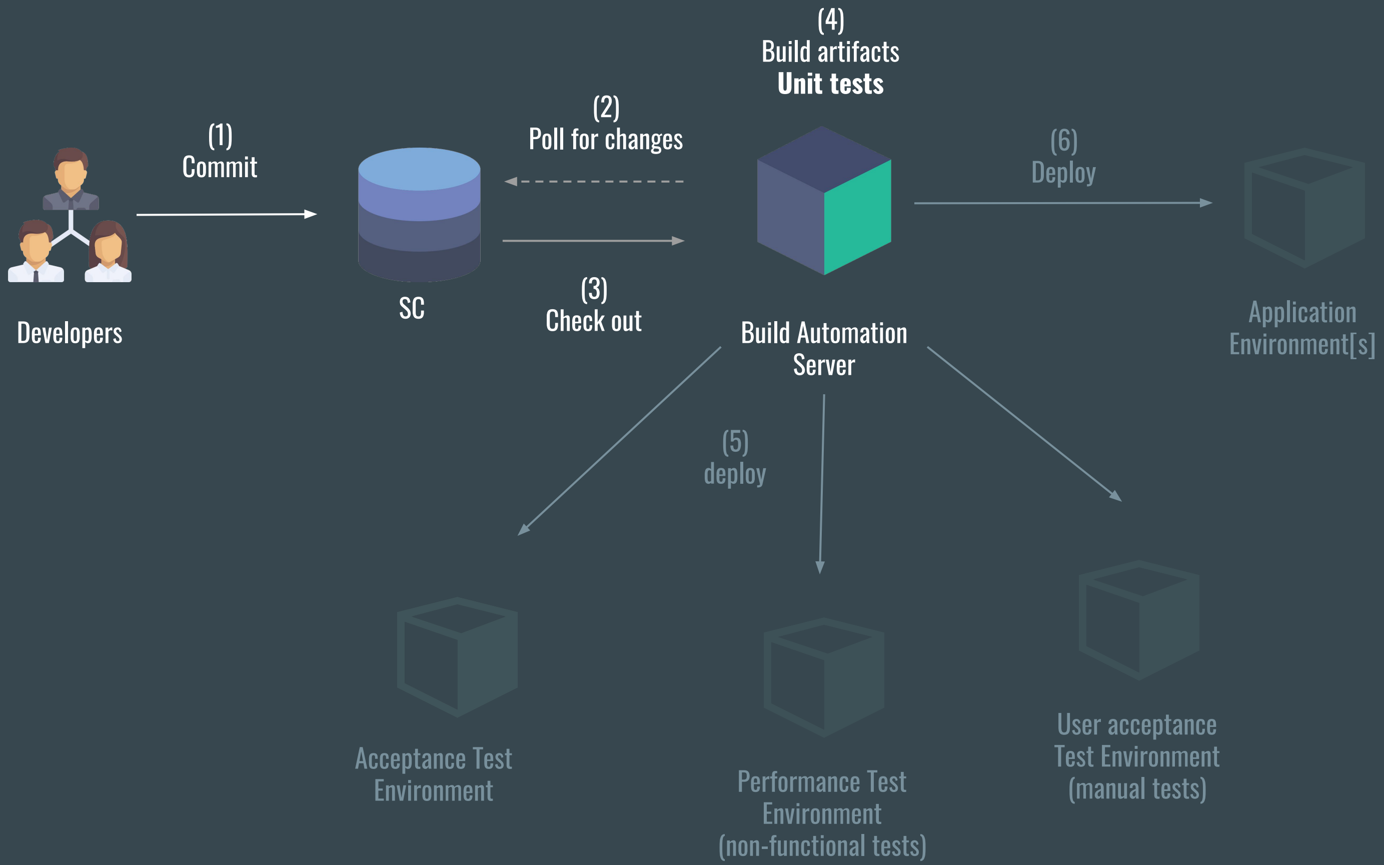
THE **BASICS**

Terminology

- Integration (CI) - Build and test
- Deployment (CD) - Deploy and integration test
- Delivery - All the way to production

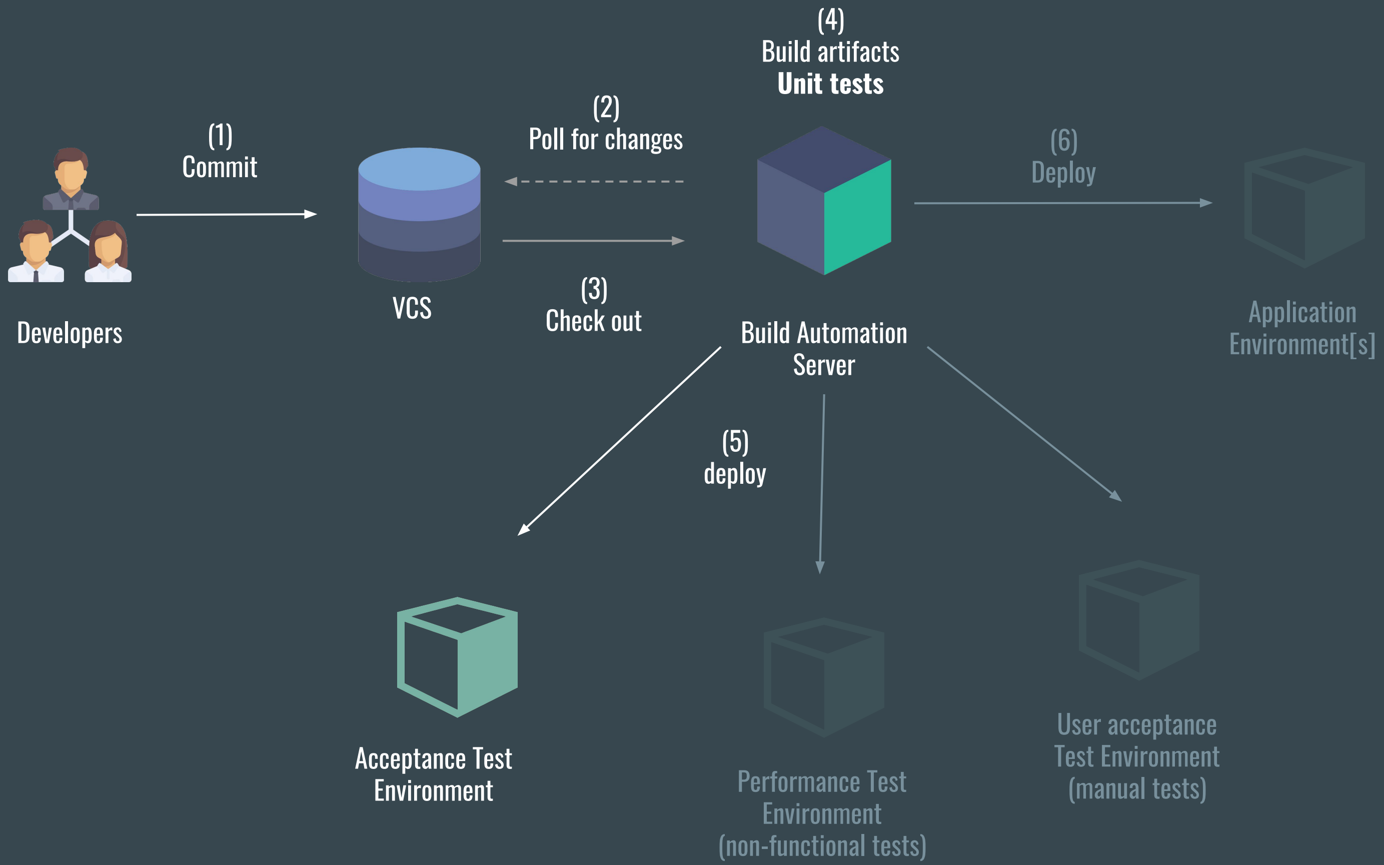


CI/CD PIPELINES *REQUIRED

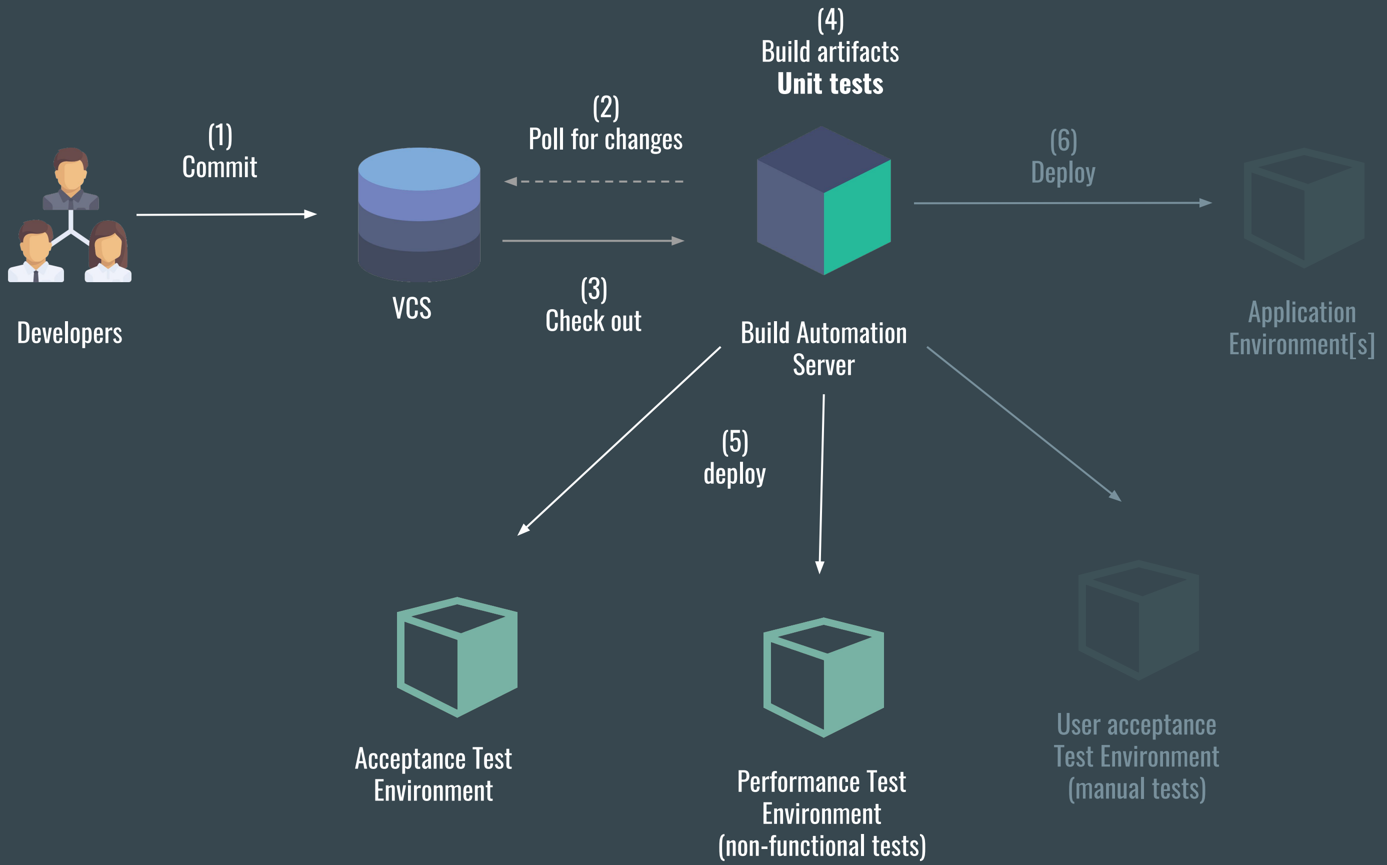


(1)
Commit
Developers
SC
(2)
Poll for changes
(3)
Check out
(4)
Build artifacts
Unit tests
Build Automation Server
(6)
Deploy
Application Environment[s]
(5)
deploy
Acceptance Test Environment
Performance Test Environment (non-functional tests)
User acceptance Test Environment (manual tests)

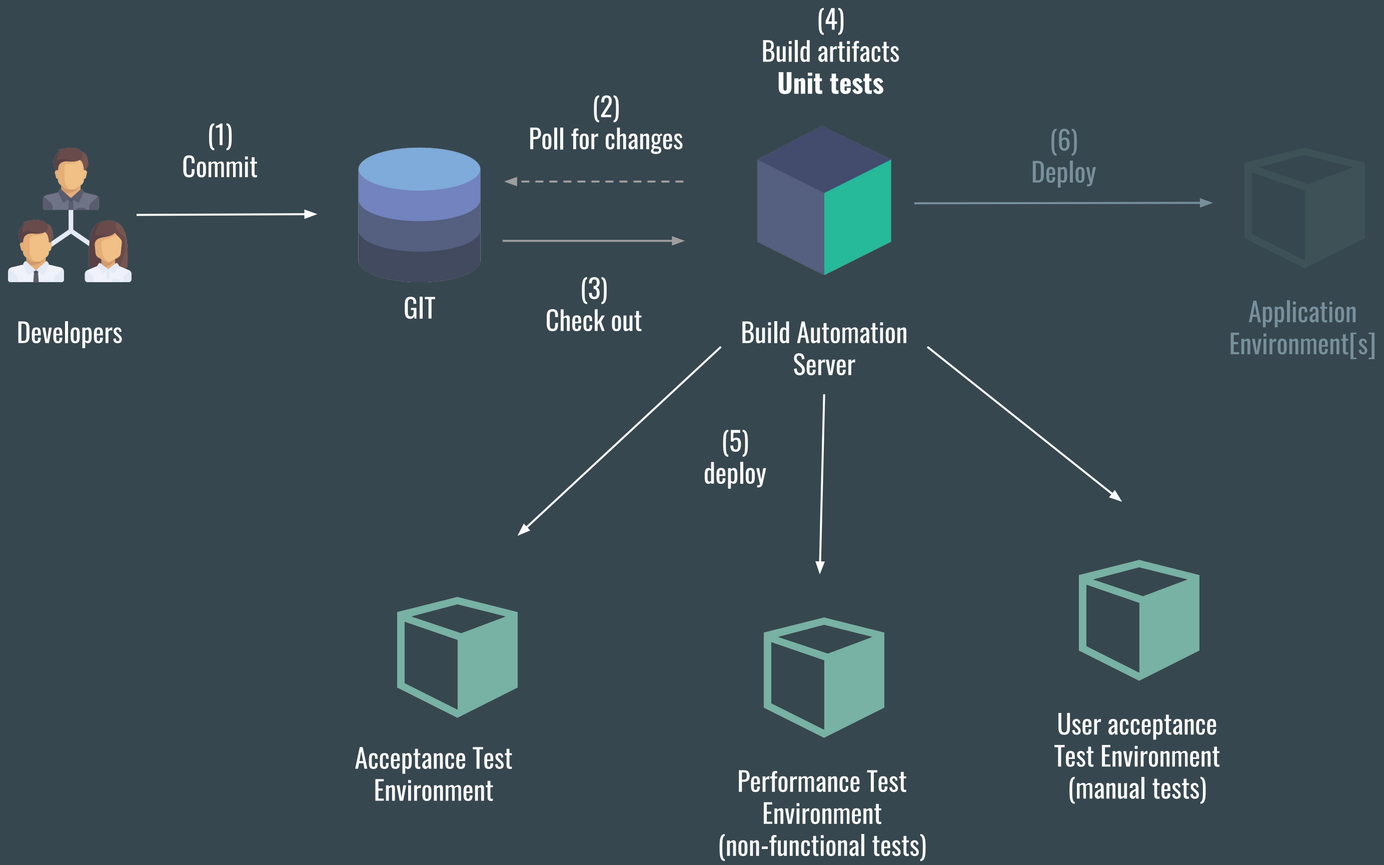
CI/CD PIPELINES *REQUIRED


(1)
Commit
(2)
Poll for changes
(3)
Check out
(4)
Build artifacts
Unit tests
(5)
deploy
(6)
Deploy
Developers
VCS
Build Automation Server
Application Environment[s]
Acceptance Test Environment
Performance Test Environment (non-functional tests)
User acceptance Test Environment (manual tests)

CI/CD PIPELINES *REQUIRED

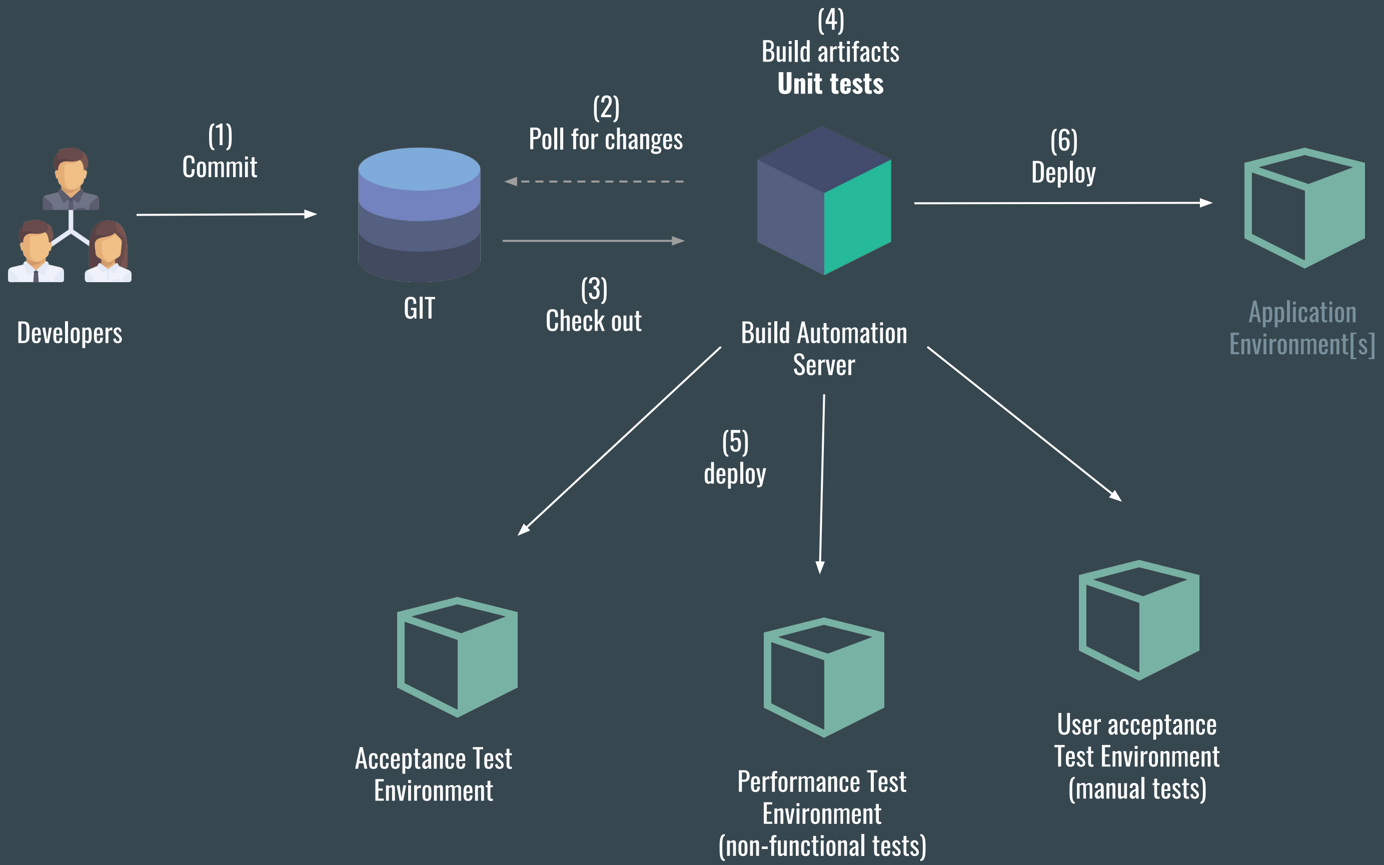

(1)
Commit
Developers
VCS
(2)
Poll for changes
(3)
Check out
(4)
Build artifacts
Unit tests
Build Automation Server
(6)
Deploy
Application Environment[s]
(5)
deploy
Acceptance Test Environment
Performance Test Environment (non-functional tests)
User acceptance Test Environment (manual tests)

CI/CD PIPELINES *REQUIRED



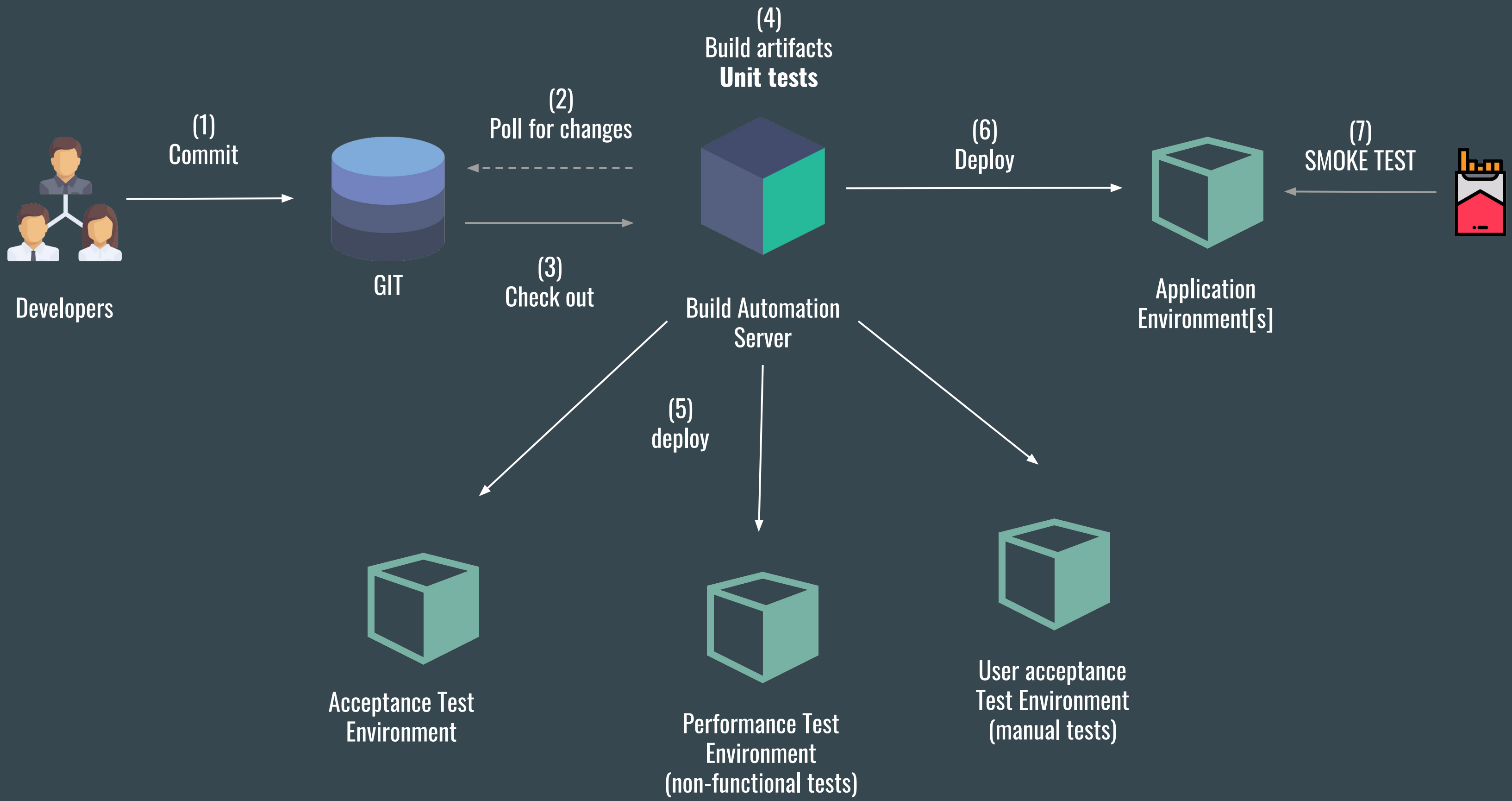
(1)
Commit
Developers
GIT
(2)
Poll for changes
(3)
Check out
(4)
Build artifacts
Unit tests
Build Automation Server
(6)
Deploy
Application Environment[s]
(5)
deploy
Acceptance Test Environment
Performance Test Environment (non-functional tests)
User acceptance Test Environment (manual tests)

CI/CD PIPELINES *REQUIRED



(1)
Commit
Developers
GIT
(2)
Poll for changes
(3)
Check out
(4)
Build artifacts
Unit tests
Build Automation Server
(6)
Deploy
Application Environment[s]
(5)
deploy
Acceptance Test Environment
Performance Test Environment (non-functional tests)
User acceptance Test Environment (manual tests)

CI/CD PIPELINES *REQUIRED


(1)
Commit
Developers
GIT
(2)
Poll for changes
(3)
Check out
(4)
Build artifacts
Unit tests
Build Automation Server
(6)
Deploy
(7)
SMOKE TEST
Application Environment[s]
(5)
deploy
Acceptance Test Environment
Performance Test Environment (non-functional tests)
User acceptance Test Environment (manual tests)

Benefits:

Automation

Reduces the risks of software releases errors that are made by manual process and environments that got deferred due to -

- Missing OS Environment variables configuration
  such as :
  - Max. open files
  - Users
  - Files and folders etc.
- OS Configuration and packages
- Middleware configuration and versioning.

Benefits:

Auditability

Know who did what, why he did it and when

Achieved that by

- Save all specifications and configuration in svn
- Provide meaningful information per commit by using the changelog messages
- Recreate environment base on revisions that got deployed

Benefits:

Repeatability

Deployment are no longer in the hand of one profissional

Repeatability

- Allows any authorized personnel to recreate environments
- Bringing “Deployment as a service” to the developers and operation team
- Catching bugs early in the flow
  - Redeploying environments in minutes with working versions - Minimizing **MTTR** $\longrightarrow$ **Mean time to repair**
- Making deployments boring and ultra reliable

Benefits Summary:

Automate deployments to achieve

- Operating system updates / packages
- Middleware & databases installation
- Artifact deployment & Dependencies management
- Data propagation
- Auditability

$\left.\right\}$ + Configuration

CI / CD DICTIONARY

CI / CD DICTIONARY

Unit Tests
Is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

Static Analysis
Static analysis is one of the leading testing techniques. A static analysis tool reviews program code, searching for application coding flaws, back doors or other malicious code that could give hackers access to critical company data or customer information.

Integration Tests
Is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

Load Tests
Is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation.

Security Tests
Is a process intended to reveal flaws in the security mechanisms of an information system that protect data and maintain functionality as intended.

Acceptance Tests
Is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

BLUE GREEN DEPLOYMENT
Is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

CANARY DEPLOYMENT
Is a pattern for rolling out releases to a subset of users or servers. The idea is to first deploy the change to a small subset of servers, test it, and then roll the change out to the rest of the servers.

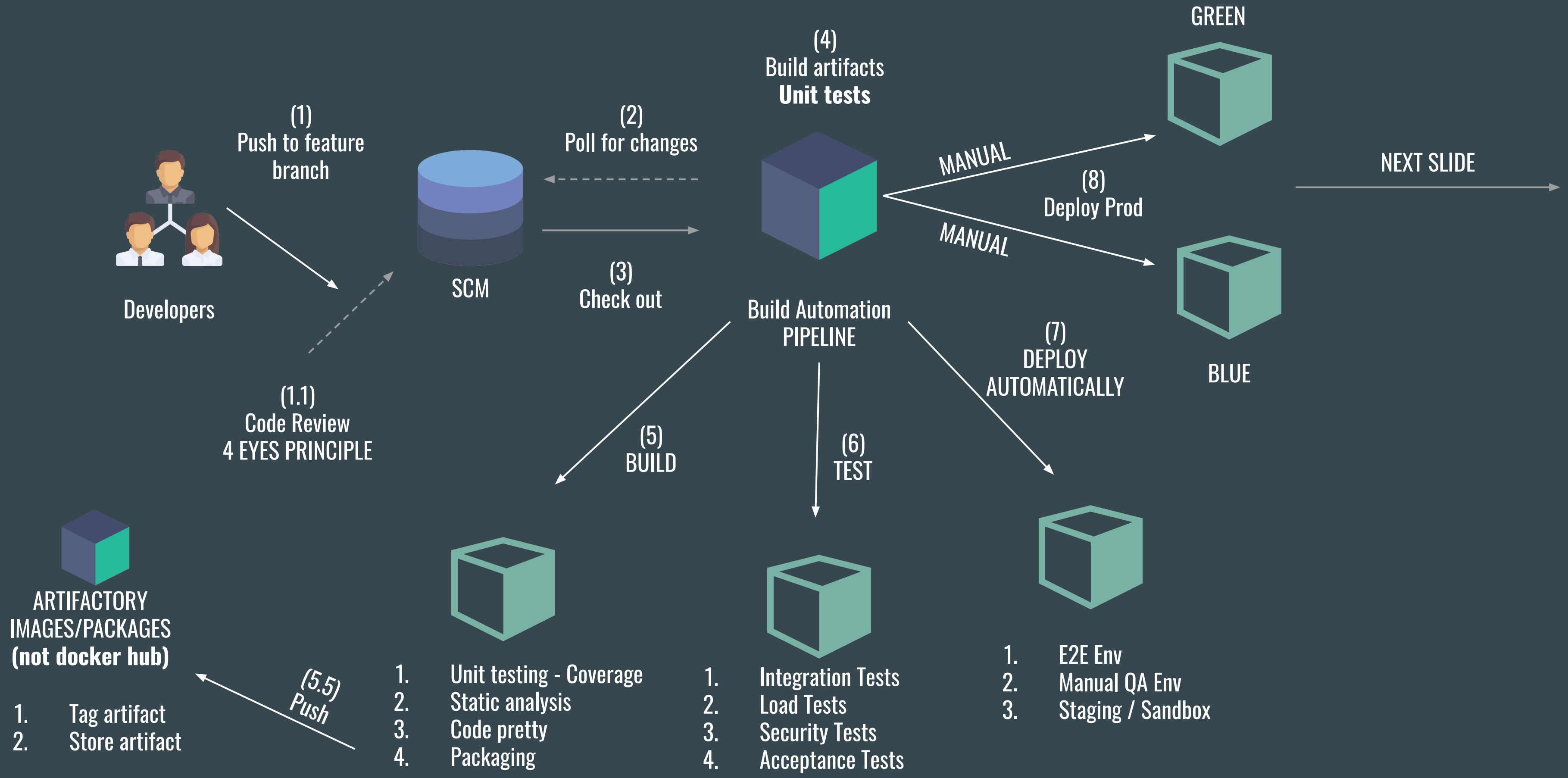
CONTAINER BASE INFRA & ARCHITECTURE

PIPELINE

SOURCE	BUILD	TEST	DEPLOY
VERSION CONTROL	COMPILATION	INTEGRATION TEST	G/B DEPLOYMENTS
BRANCHING	UNIT TESTS	LOAD TESTS	CANARY DEPLOYMENTS
CODE REVIEW	STATIC ANALYSIS	SECURITY TESTS	MONITORING
	CODE PRETTY	ACCEPTANCE - TESTS	MEASURING
	REPORTING		VALIDATION
	PACKAGING		RELEASE TO THE WILD
	(DOCKER IMAGE)		

- PIPELINE EXAMPLE -

CI/CD PIPELINE TBD



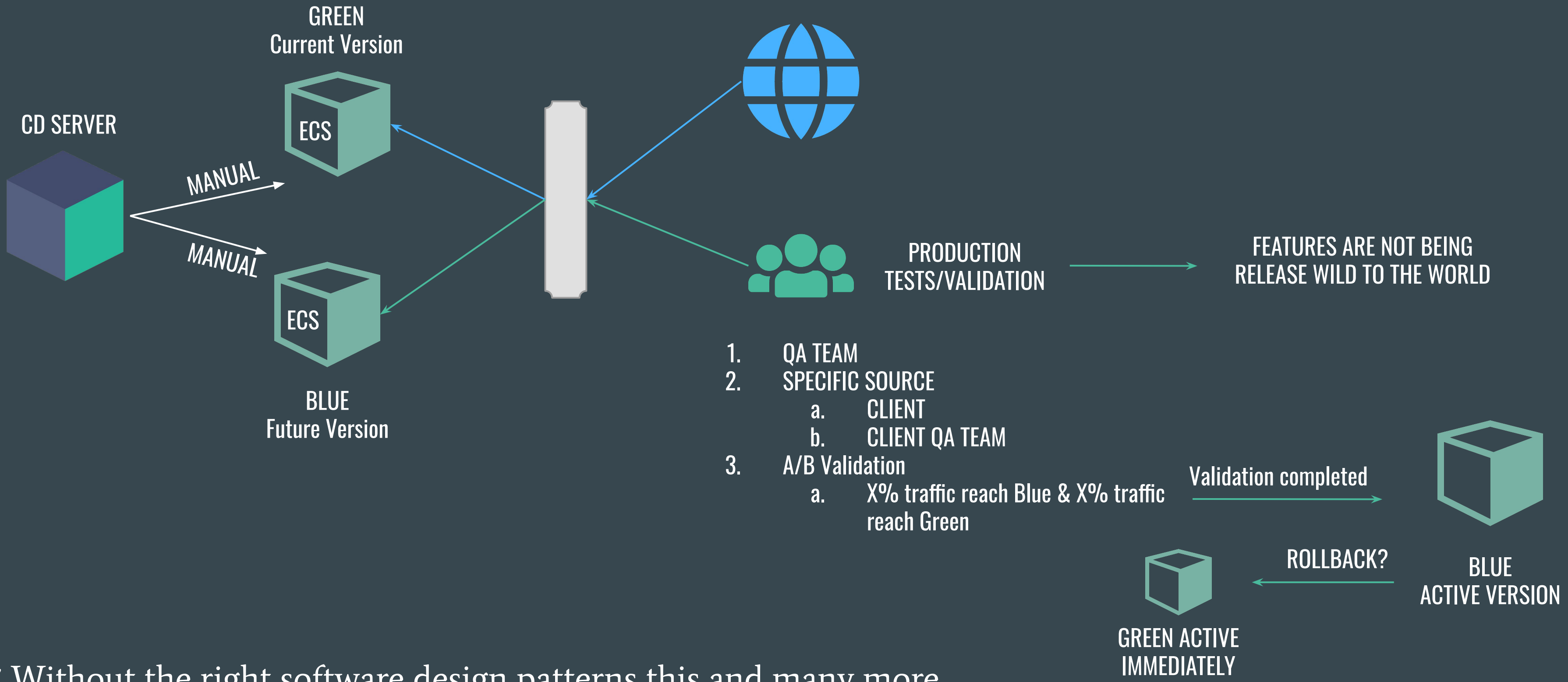
1. Tag artifact
2. Store artifact

1. Unit testing - Coverage
2. Static analysis
3. Code pretty
4. Packaging

1. Integration Tests
2. Load Tests
3. Security Tests
4. Acceptance Tests

1. E2E Env
2. Manual QA Env
3. Staging / Sandbox

CD PIPELINE PRODUCTION



1. QA TEAM
2. SPECIFIC SOURCE
    a. CLIENT
    b. CLIENT QA TEAM
3. A/B Validation
    a. X% traffic reach Blue & X% traffic reach Green

** Without the right software design patterns this and many more functionality such as rolling updates will not be achievable

CI / CD DICTIONARY - EXTENDING

PACKER

is a tool for building images for cloud platforms, virtual machines, containers, and more from a single source configuration. Packer is a powerful and full-featured tool to create cloud images and application packages

- JENKINS -

WHAT IS JENKINS

Jenkins

Is a self-contained, open source automation service which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

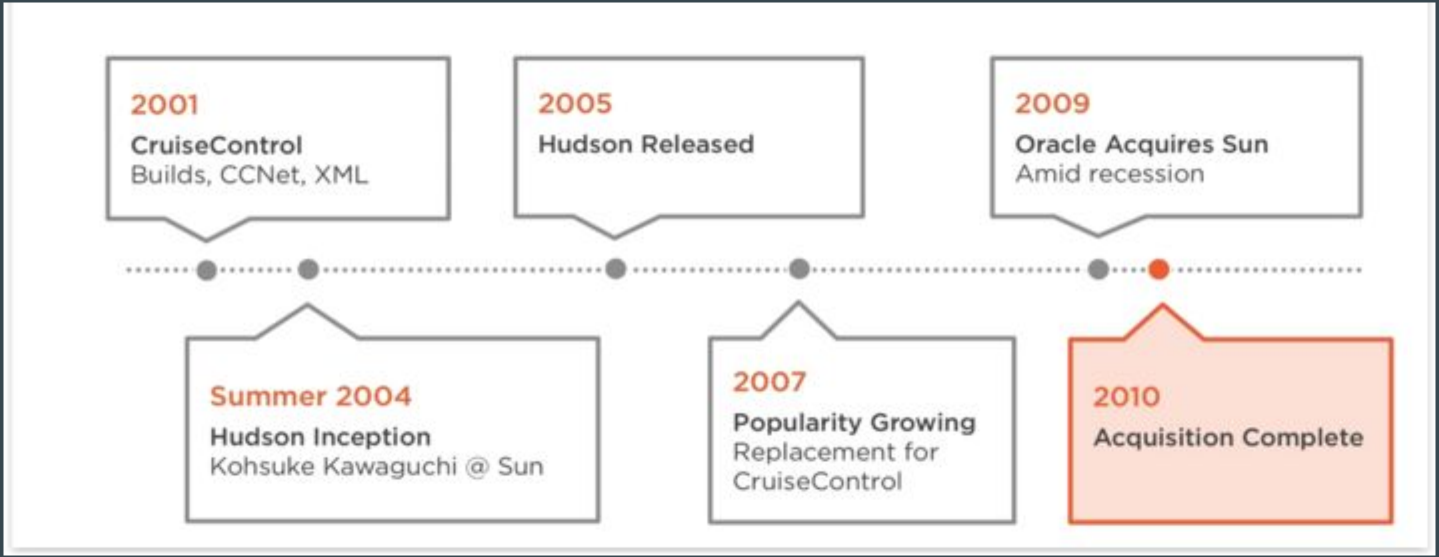
The tool offers a simple way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines (scripts), as well as automating other routine development tasks.

WHAT IS JENKINS

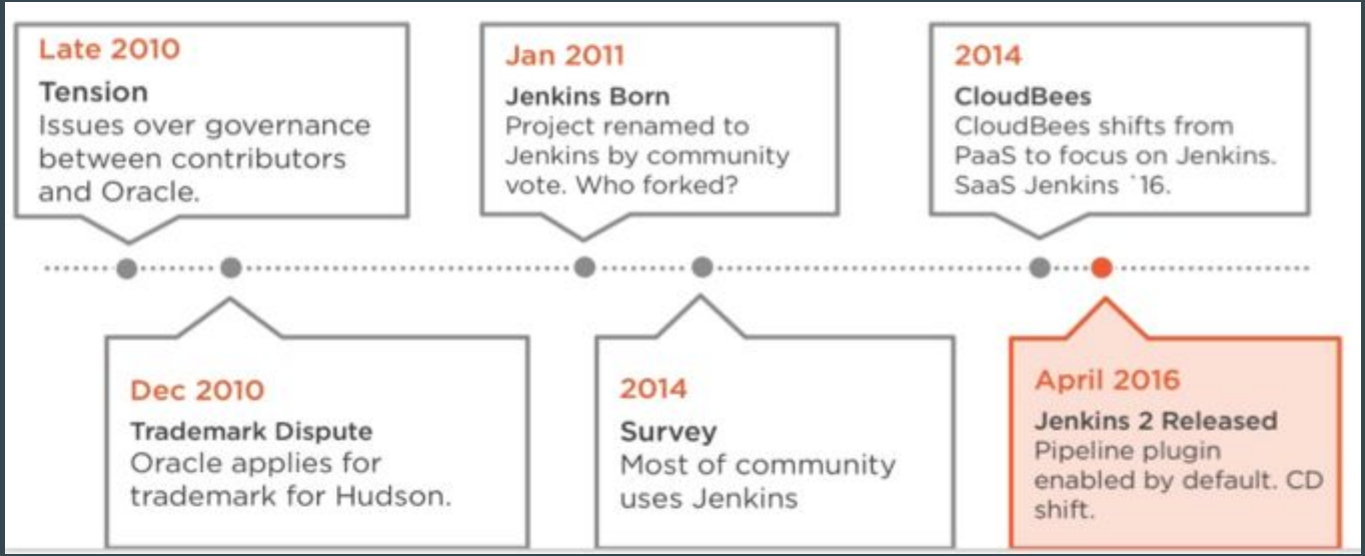
Jenkins

- Branched from Hudson
- Java based Continuous Build System
- Runs in servlet container Tomcat
- Supported by over 1600+ plugins and 1000+ community contributors (https://plugins.jenkins.io/)
- Under development since 2005

Some History and Timeline



Some History and Timeline



Late 2010
Tension
Issues over governance between contributors and Oracle.

Dec 2010
Trademark Dispute
Oracle applies for trademark for Hudson.

Jan 2011
Jenkins Born
Project renamed to Jenkins by community vote. Who forked?

2014
Survey
Most of community uses Jenkins

2014
CloudBees
CloudBees shifts from PaaS to focus on Jenkins. SaaS Jenkins '16.

April 2016
Jenkins 2 Released
Pipeline plugin enabled by default. CD shift.

WHAT IS JENKINS

Jenkins Installation options

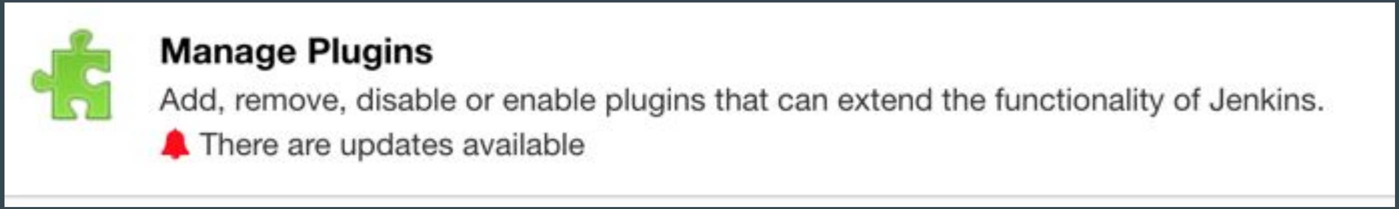
Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed:

- Master direct Installation
- JAR/WAR BASE
- Docker BASE - **We will use the Docker type installation and configuration**

WHAT IS JENKINS

WHY Jenkins?

Jenkins is a highly configurable system by itself
The additional community developed plugins provide even more flexibility
By combining Jenkins with Ant, Gradle, or other Build Automation tools, the possibilities are limitless



Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

There are updates available

- JENKINS PLUGINS -

JENKINS PLUGINS

Most Commonly Used Plugins for Jenkins

- Blue Ocean UI Plugin
- GIT
- GITHUB
- BUILDS
- MSBUILD
- MAVEN
- TESTING
- SELENIUM
- DOCKER

OCEAN UI PLUGIN

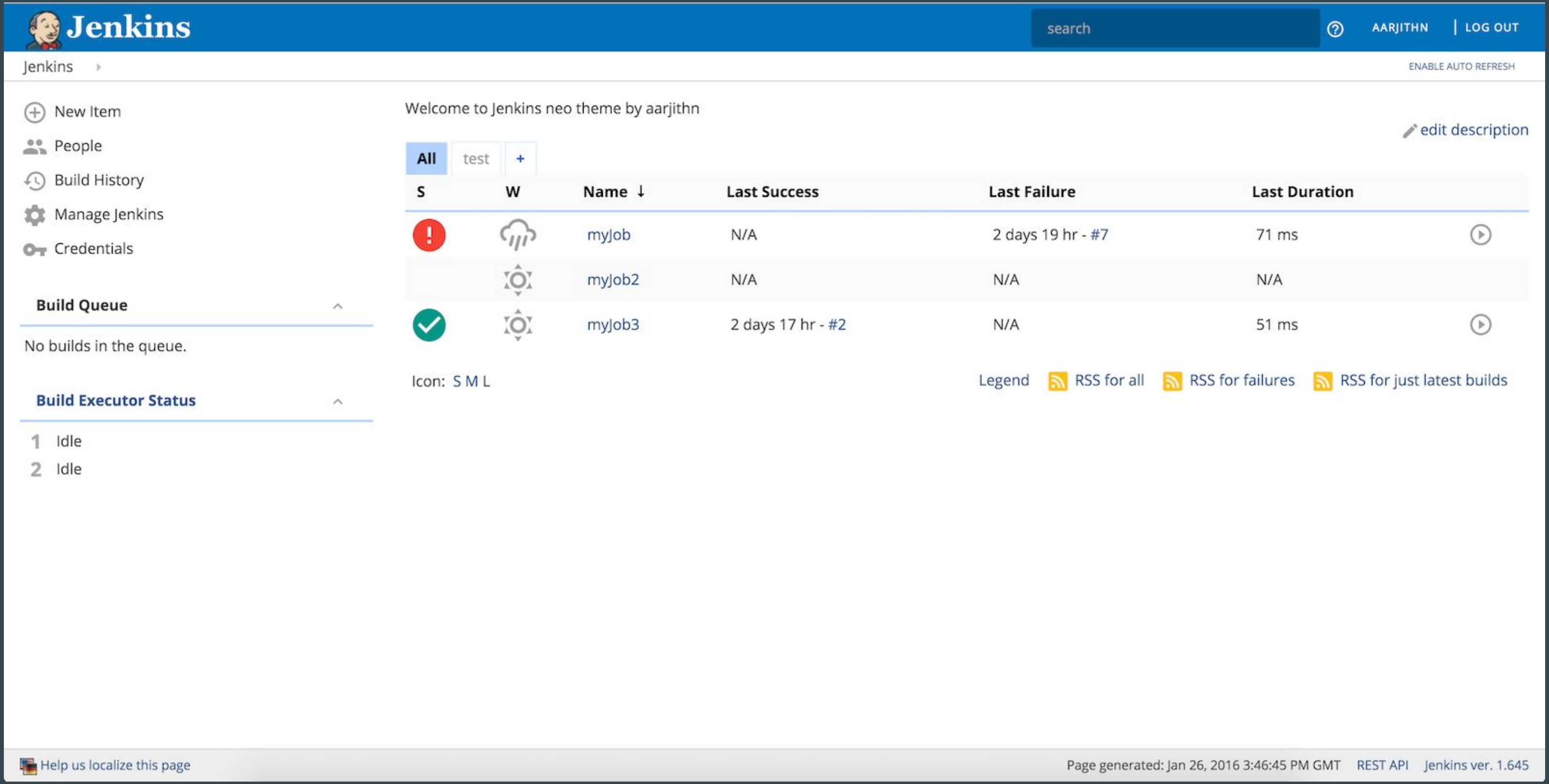
Sophisticated visualizations of CD pipelines, allowing for fast and intuitive comprehension of software pipeline status.

Pipeline editor that makes automating CD pipelines approachable by guiding the user through an intuitive and visual process to create a pipeline.

Personalization of the Jenkins UI to suit the role-based needs of each member of the DevOps team.
Pinpoint precision when intervention is needed and/or issues arise. The Blue Ocean UI shows where in the pipeline attention is needed, facilitating exception handling and increasing productivity.

Native integration for branch and pull requests enables maximum developer productivity when collaborating on code with others in GitHub and Bitbucket.

JENKINS PLUGINS



Jenkins | search | AARJITHN | LOG OUT

Jenkins | ENABLE AUTO REFRESH

New Item
People
Build History
Manage Jenkins
Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

Welcome to Jenkins neo theme by aarjithn

edit description

All | test | +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
!		myJob	N/A	2 days 19 hr - #7	71 ms	▶
		myJob2	N/A	N/A	N/A	
✓		myJob3	2 days 17 hr - #2	N/A	51 ms	▶

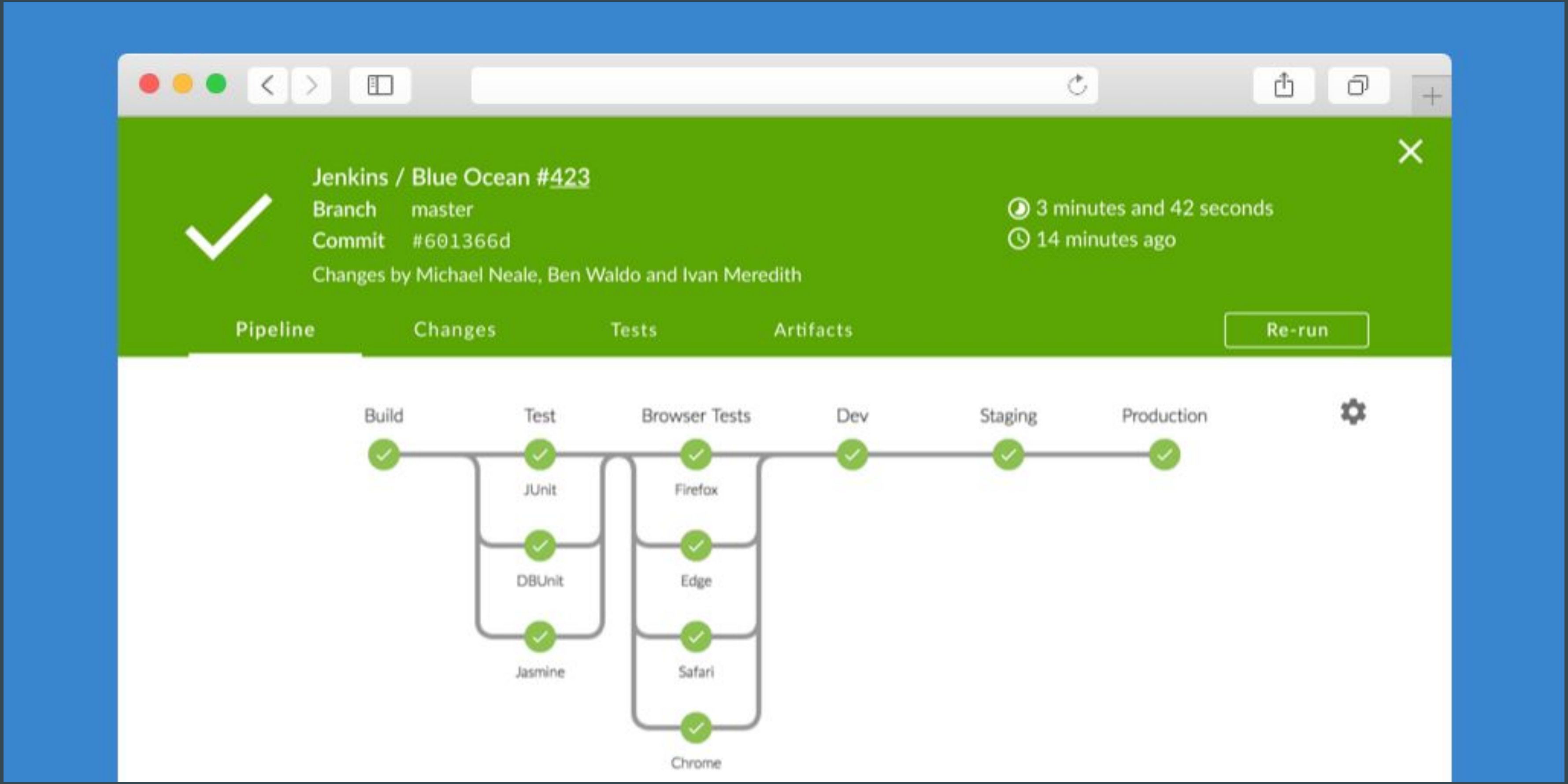
Icon: S M L

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

Help us localize this page

Page generated: Jan 26, 2016 3:46:45 PM GMT | REST API | Jenkins ver. 1.645

JENKINS PLUGINS



Jenkins / Blue Ocean #423
Branch master
Commit #601366d
Changes by Michael Neale, Ben Waldo and Ivan Meredith

3 minutes and 42 seconds
14 minutes ago

GIT

Git is a version control system created by Linus Torvalds in 2005 for development of the Linux kernel, having multiple advantages over the other systems available, it stores file changes more efficiently and ensures file integrity better.

Don’t know GIT?

10 min exercise - https://try.github.io/levels/1/challenges/1

JENKINS PLUGINS

GITHUB

- Create hyperlinks between your Jenkins projects and GitHub
- Trigger a job when you push to the repository by froking HTTP POSTs from post-receive hook and optionally auto-managing the hook setup.
- Report build status result back to github as Commit Status (documented on SO)
- Base features for other plugins

BUILD

Once a project is successfully created in Jenkins, all future builds are automatic
Jenkins executes the build in an executor.

By default, Jenkins gives one executor per core on the build server.

Jenkins also has the concept of slave build servers
Useful for building on different architectures and Distribution of load

MSBUILD

This plugin allows you to use MSBuild to build .NET projects.

To use this plugin, specify the location directory of MSBuild.exe on Jenkin's configuration page.
Then, on your project configuration page, specify the name of the build file (.proj or .sln) and any command line arguments you want to pass in.
The files are compiled to the directory where Visual Studio would put them as well.

JENKINS PLUGINS

MAVEN

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

This plugin provides an advanced integration for Maven.

- Automatic configuration of reporting plugins (Junit, Findbugs, ...)
- Automatic triggering across jobs based on SNAPSHOTs published/consumed
- Incremental build - only build changed modules
- Build modules in parallel on multiple executors/nodes
- Post build deployment of binaries only if the project succeeded and all tests passed

JENKINS PLUGINS

SELENUIM

Automates browsers..... And that's it :-)

DOCKER

Docker plugin allows to use a docker host to dynamically provision build agents,
run a single build, then tear-down agent.

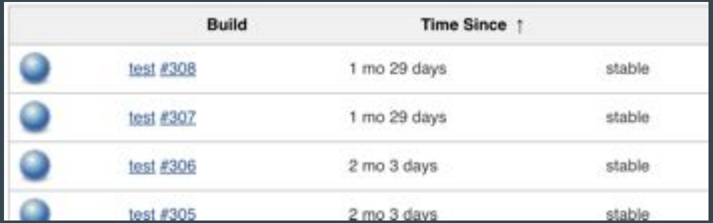
Optionally, the container can be committed, so that (for example) manual QA could be performed by the container being imported into a local docker provider, and run from there.

- JENKINS FEATURES -

JENKINS FEATURES

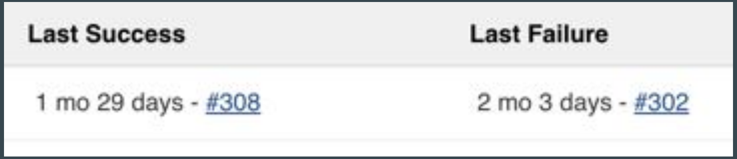
REPORTING AND NOTIFICATIONS

Jenkins comes with basic reporting features
Keeping track of build status



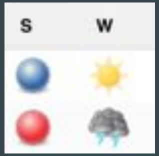
	Build	Time Since ↑	
●	test #308	1 mo 29 days	stable
●	test #307	1 mo 29 days	stable
●	test #306	2 mo 3 days	stable
●	test #305	2 mo 3 days	stable

Last success and failure



Last Success	Last Failure
1 mo 29 days - #308	2 mo 3 days - #302

"Weather" – Build trend



S	W
🔵	☀
🔴	⛈

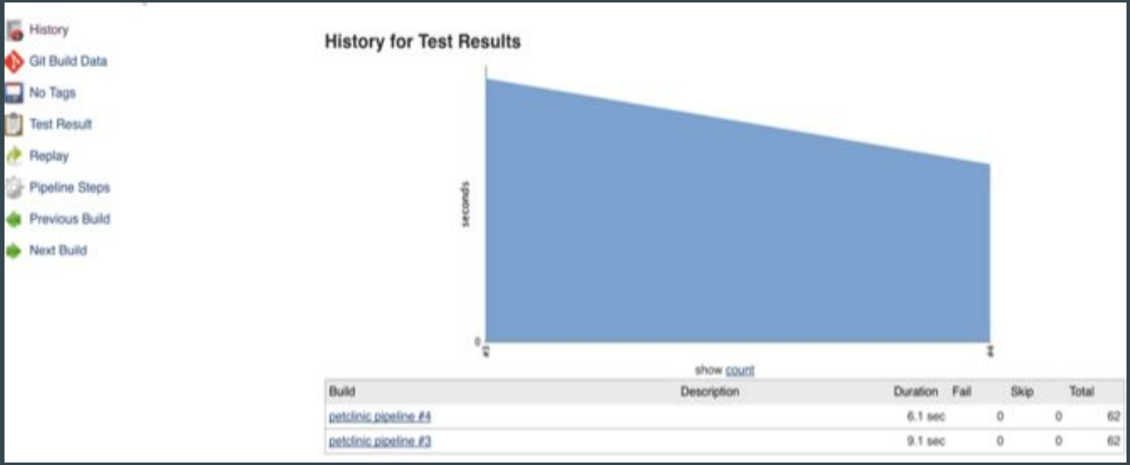
JENKINS FEATURES

REPORTING AND NOTIFICATIONS

The basic reporting can be greatly enhanced with the use of pre-build plugins such as

- Unit test coverage
- Test result trending
- Findbugs, Checkstyle, PMD

REPORTING AND NOTIFICATIONS

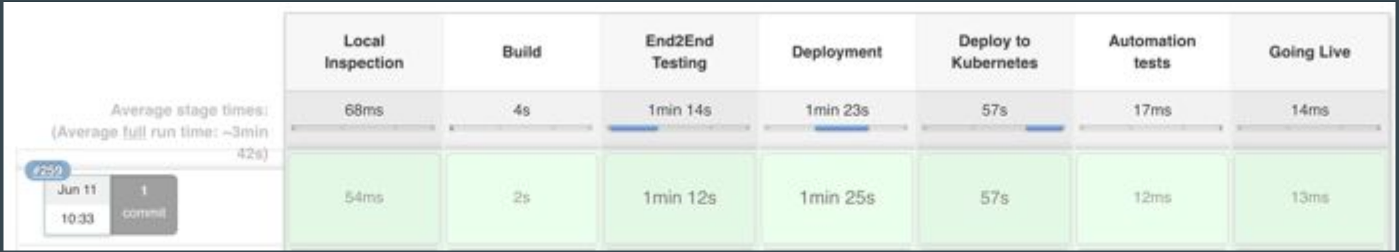


Build	Description	Duration	Fail	Skip	Total
petclinic pipeline #4		6.1 sec	0	0	62
petclinic pipeline #3		9.1 sec	0	0	62

JENKINS FEATURES

PIPELINES (AS A CODE - Jenkins The Devops Way)

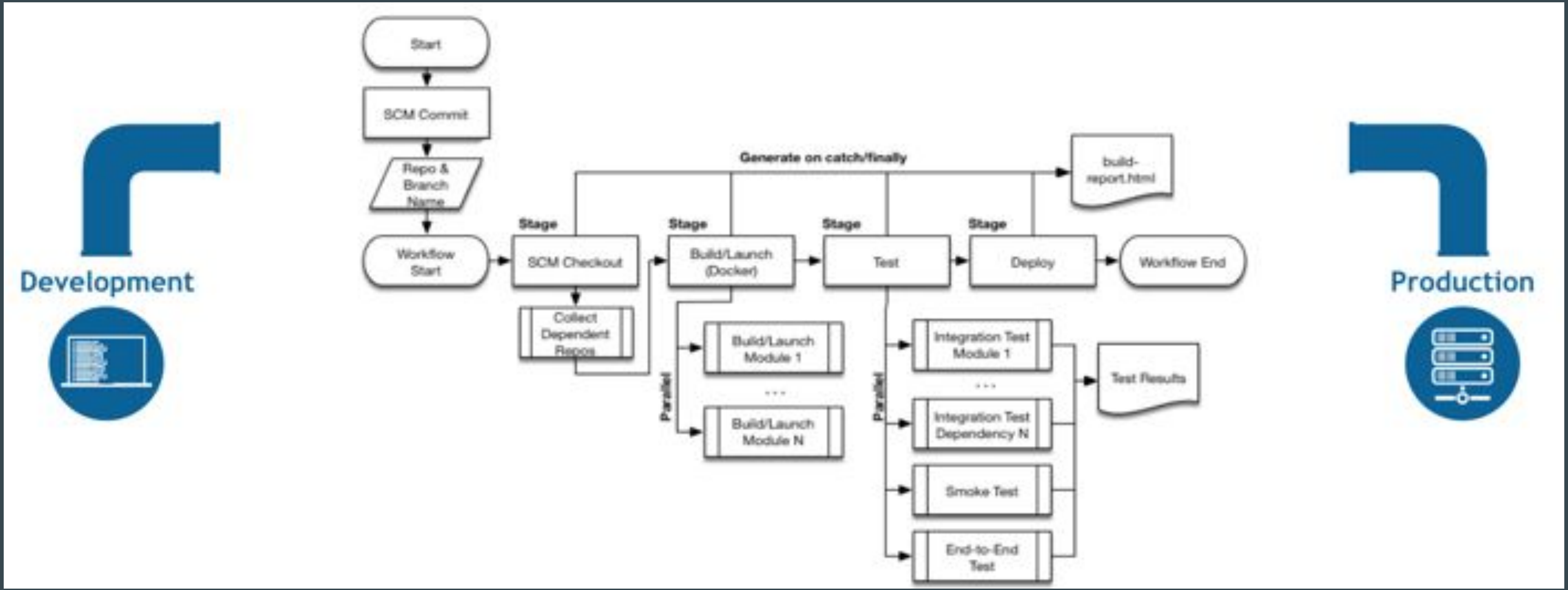
- Pipeline as Code. You could supervise and keep the changes stored in GIT
- Easily define simple and complex pipelines through the DSL in a Jenkinsfile.
- Pipeline as code provides a common language to help teams (e.g. Dev and Ops) work together.
- Easily share pipelines between teams by storing common "steps" in shared repositories.



	Local Inspection	Build	End2End Testing	Deployment	Deploy to Kubernetes	Automation tests	Going Live
Average stage times: (Average full run time: ~3min 42s)	68ms	4s	1min 14s	1min 23s	57s	17ms	14ms
[illegible] Jun 11 10:33 1 commit	54ms	2s	1min 12s	1min 25s	57s	12ms	13ms

JENKINS FEATURES

PIPELINES

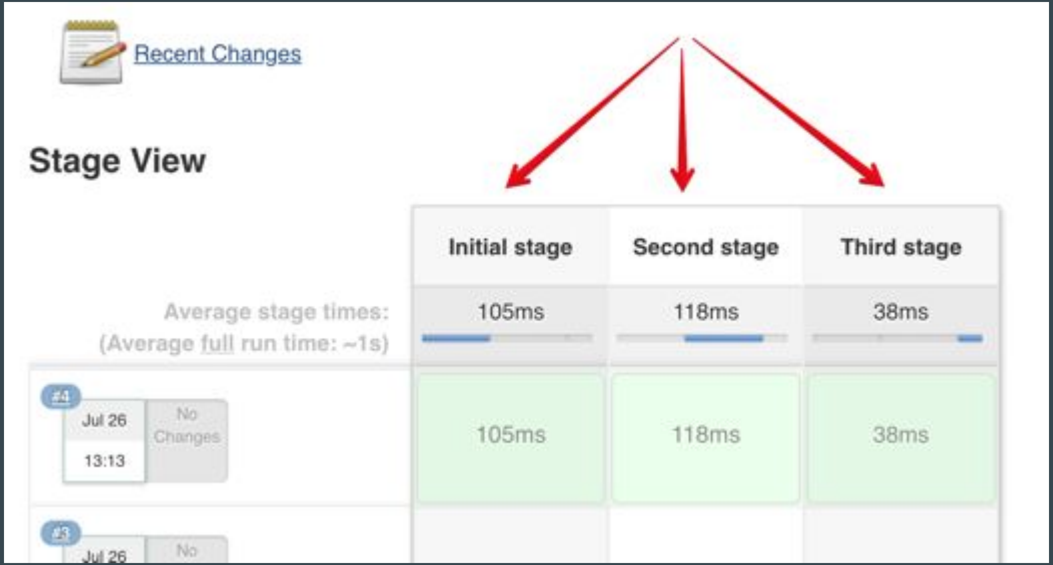


PIPELINES (AS A CODE - Jenkins The Devops Way)

Example

```
node {
//My first test pipeline
    stage('Initial stage')
        {
            echo 'Thanks John Bryce'
        }
    stage('Second stage')
        {
            def text = 'John'
            def text2 = 'Bryce'
            echo "${text} ${text2} has reached the second stage"
        }
    stage('Third stage')
        {
            println 'Thanks for your patience'
        }
}
```

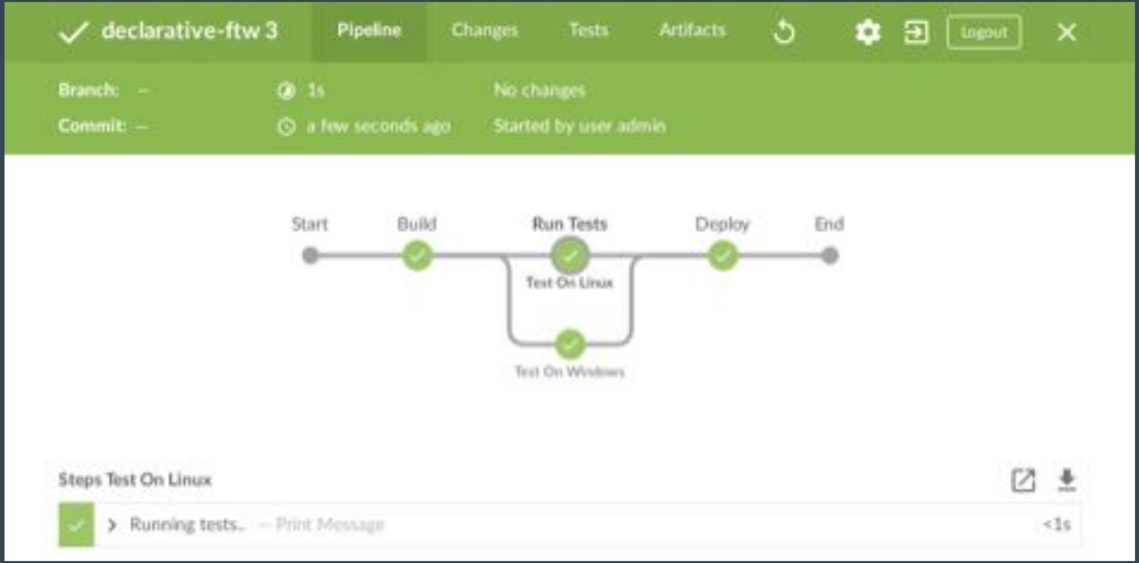
PIPELINES (AS A CODE - Jenkins The Devops Way)



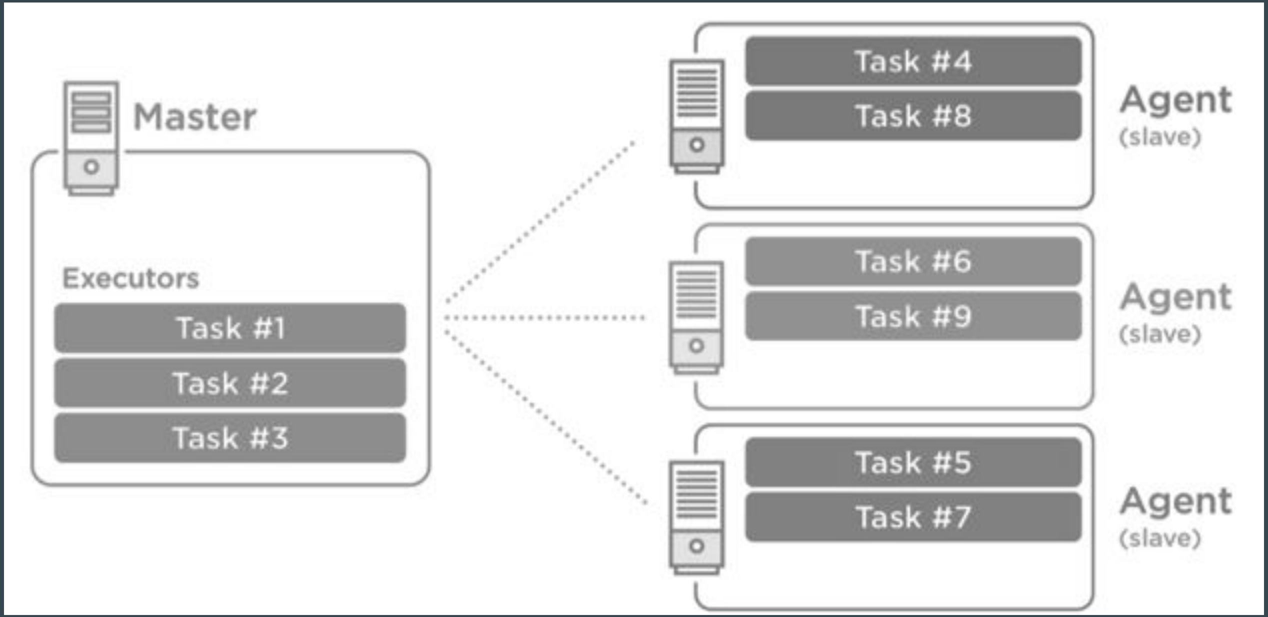
	Initial stage	Second stage	Third stage
Average stage times: (Average full run time: ~1s)	105ms	118ms	38ms
#4 Jul 26 13:13 No Changes	105ms	118ms	38ms
#3 Jul 26 No			

PIPELINES PARALLEL

In some cases there is a need to run the scripts in parallel. For instance we have a test step to run on the different nodes with different OS and in a sake of saving the time resource we could write the parallel commands to be triggered simultaneously:

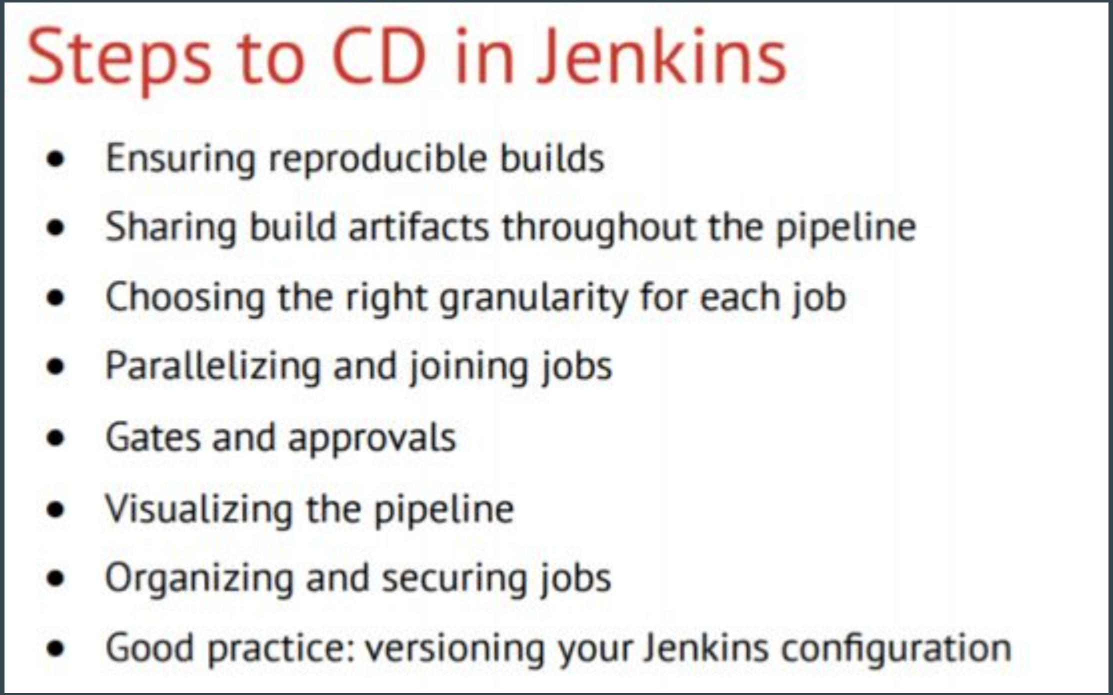


MASTER - AGENT MODEL



- STEPS TO CD IN JENKINS -

STEPS TO CD IN JENKINS



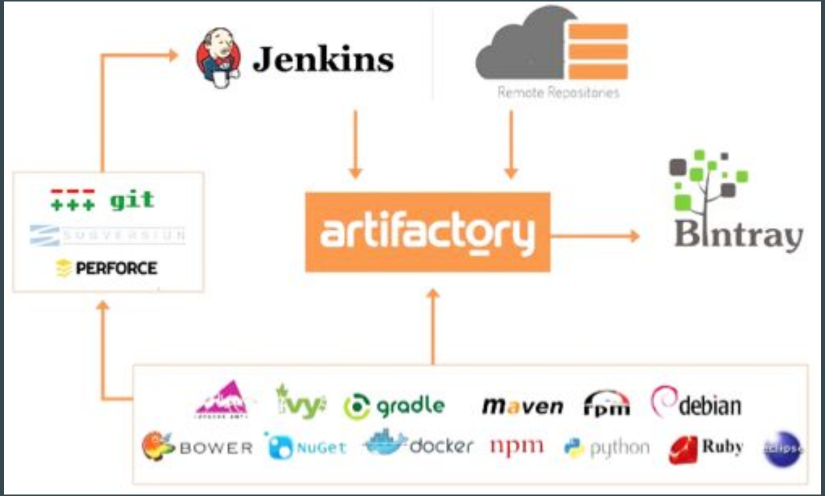
Steps to CD in Jenkins

- Ensuring reproducible builds
- Sharing build artifacts throughout the pipeline
- Choosing the right granularity for each job
- Parallelizing and joining jobs
- Gates and approvals
- Visualizing the pipeline
- Organizing and securing jobs
- Good practice: versioning your Jenkins configuration

- ARTIFACTS REPOSITORY -

ARTIFACTS REPOSITORY

CI/CD stages supposed to have a storage where all the versioned product versions would be stored for the deployment ether whilst the “Deploy and testing environment” stage or “Deploy to production environment” stage.



ARTIFACTS REPOSITORY

NEXUS

There are cases when a company couldn’t afford to keep the artifacts for the delivery or deployment on the external cloud services.

The security policies tend them to move the storages inside the secured network. Using a private antifactory managers may solve the issue.

Sonatype Nexus Repository is the installable repository manager with the wide list of features and sophisticated API to be integrated into your infrastructure

ARTIFACTS REPOSITORY

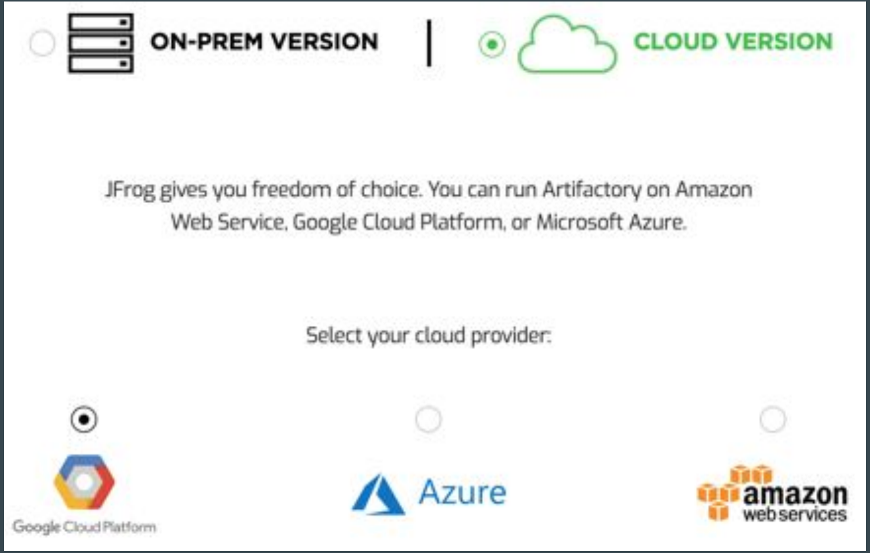
NEXUS - FEATURES

- Manage components, build artifacts, and release candidates in one central location.
- Understand component security, license, and quality issues.
- Modernize software development with intelligent staging and release functionality.
- Scale DevOps delivery with high availability and active/active clustering.
- Sleep comfortably with world-class support and training.
- Store and distribute Maven/Java, npm, NuGet, RubyGems, Docker, P2, OBR, APT and YUM and more.
- Manage components from dev through delivery: binaries, containers, assemblies, and finished goods.
- Awesome support for the Java Virtual Machine (JVM) ecosystem, including Gradle, Ant, Maven, and Ivy.
- Integrated with popular tools like Eclipse, IntelliJ, Hudson, Jenkins, Puppet, Chef, Docker, and more.

ARTIFACTS REPOSITORY

JFROG

Artifactory is an artifact repository manager, which is entirely technology agnostic and fully supports software created in any language or using any tool



[ ] ON-PREM VERSION | [✓ CLOUD VERSION]

JFrog gives you freedom of choice. You can run Artifactory on Amazon Web Service, Google Cloud Platform, or Microsoft Azure.

Select your cloud provider:

[✓ Google Cloud Platform] [ ] Azure [ ] amazon web services

JFROG - FEATURES

The tool is designed to integrate with the majority of continuous integration and delivery tools,
to provide an end to end automated solution for tracking artifacts from development to production.

JFrog Artifactory is intended for use both by developers and DevOps teams as a whole.
It serves as a singles access point that organizes all of the resources and removes the associated complications.
Simultaneously it enables the operations staff to efficiently manage the continual flow of code from each developer’s machine to the organization’s production environment.

- INSTALLING JENKINS2 - THE DOCKER WAY

JENKINS INSTALLATION REPOSITORY

READY

```
docker run -tid -p 8080:8080 -p 5000:5000 --volume jenkins_home:/var/jenkins_home --volume
/var/run/docker.sock:/var/run/docker.sock --name jenkins-master yanivomc/jenkins-master:2.165
```

SET

```
docker exec jenkins-master cat /var/jenkins_home/secrets/initialAdminPassword
```

GO

Browse: **http://localhost:8080/**
1. Paste the Initial password from previous step
2. Install “Recommended plugins”
3. Create your own User

DONE :-)

- MAKING A CHANGE -

Module 1 - JENKINS THEORY

●●●

Thank you and see you on Module 2

Provided by DevopShift for JB & Presented by DevopShift CEO Yaniv Cohen

- MAKING A CHANGE -

●●●

Module 2 - JENKINS HANDS ON

Provided by DevopShift for JB & Presented by DevopShift CEO Yaniv Cohen

AGENDA

- Building NODE JS Application
  - LAB
- IAC Automation
- Jenkins DSL
  - LAB
- Jenkins PIPELINES
  - LAB
- Jenkins integration
  - LAB

The manual way

Our Devops way is to write everything in a code but for the purpose of this demo we will see how to create a Jenkins job / Build manually.

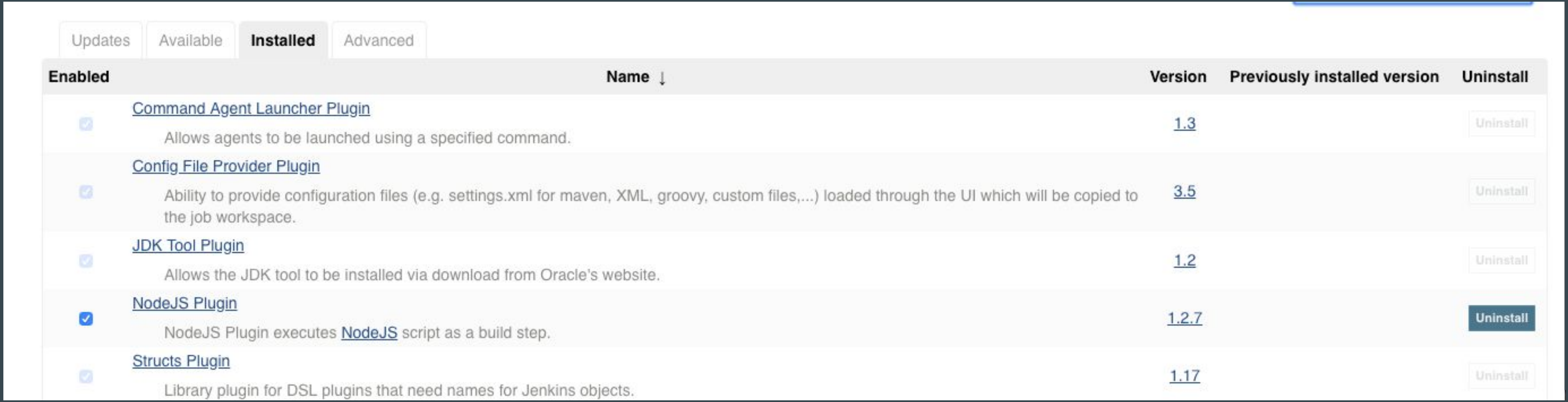
Once completed we will continue and write it as a code.

- REVIEW THE PROJECT CODE -

https://github.com/yanivomc/docker-cicd

CONFIGURING JENKINS

- INSTALL NODEJS Plugin


Updates | Available | Installed | Advanced

Enabled	Name ↓	Version	Previously installed version	Uninstall
✓	Command Agent Launcher Plugin Allows agents to be launched using a specified command.	1.3		Uninstall
✓	Config File Provider Plugin Ability to provide configuration files (e.g. settings.xml for maven, XML, groovy, custom files,...) loaded through the UI which will be copied to the job workspace.	3.5		Uninstall
✓	JDK Tool Plugin Allows the JDK tool to be installed via download from Oracle's website.	1.2		Uninstall
✓	NodeJS Plugin NodeJS Plugin executes NodeJS script as a build step.	1.2.7		Uninstall
✓	Structs Plugin Library plugin for DSL plugins that need names for Jenkins objects.	1.17		Uninstall

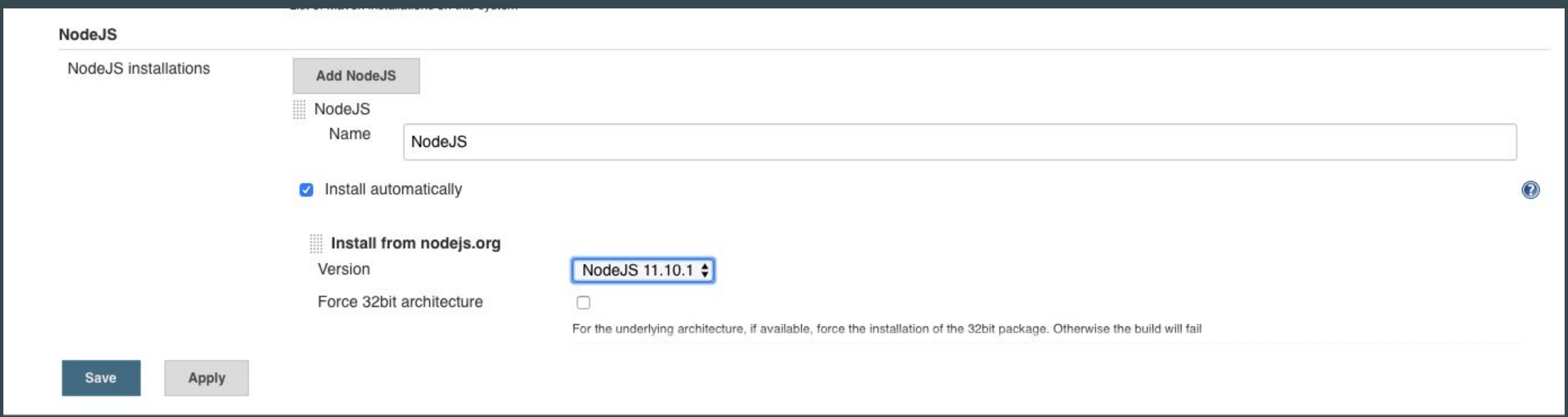
CONFIGURING JENKINS

- INSTALL NODEJS Plugin
- GLOBAL TOOL CONFIGURATION
  - Set NODEJS Installation name and Version
- CREATE NEW JOB BASE ON OUR DEMO REPO
  - Name: nodeJS-demo
  - Type: FreeStyle Project
  - Source Code Management: GIT
    - Repo URL: https://github.com/yanivomc/docker-cicd
  - Build
    - Execute Shell command
      - npm install
  - RUN JOB

BUILDING OUR FIRST APPLICATION

GLOBAL TOOL CONFIGURATION

- Configure NODEJS installation and version



NodeJS

NodeJS installations

Add NodeJS

NodeJS

Name: NodeJS

[✓] Install automatically

Install from nodejs.org

Version: NodeJS 11.10.1

Force 32bit architecture: [ ]

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

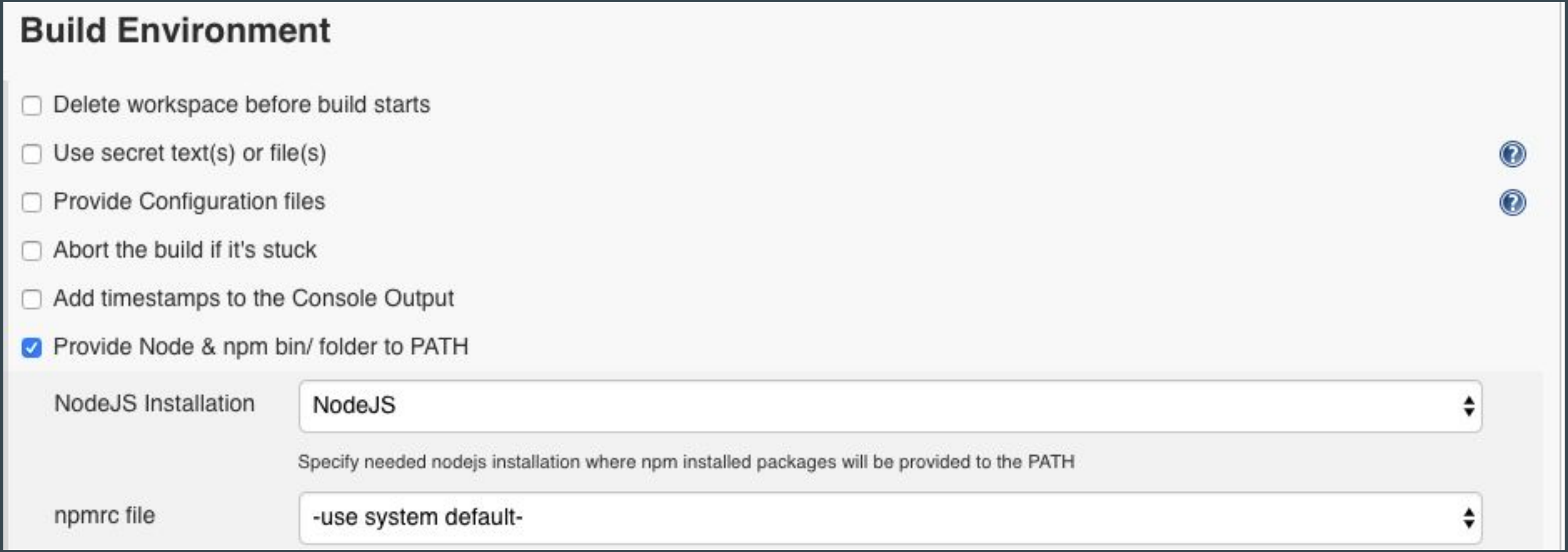
Save | Apply

BUILDING OUR FIRST APPLICATION

CREATE NEW JOB

- Add FreeStyle Job
- Configure REPO
- Configure Build Env
- Configure Build Execute command

BUILDING OUR FIRST APPLICATION



Build Environment

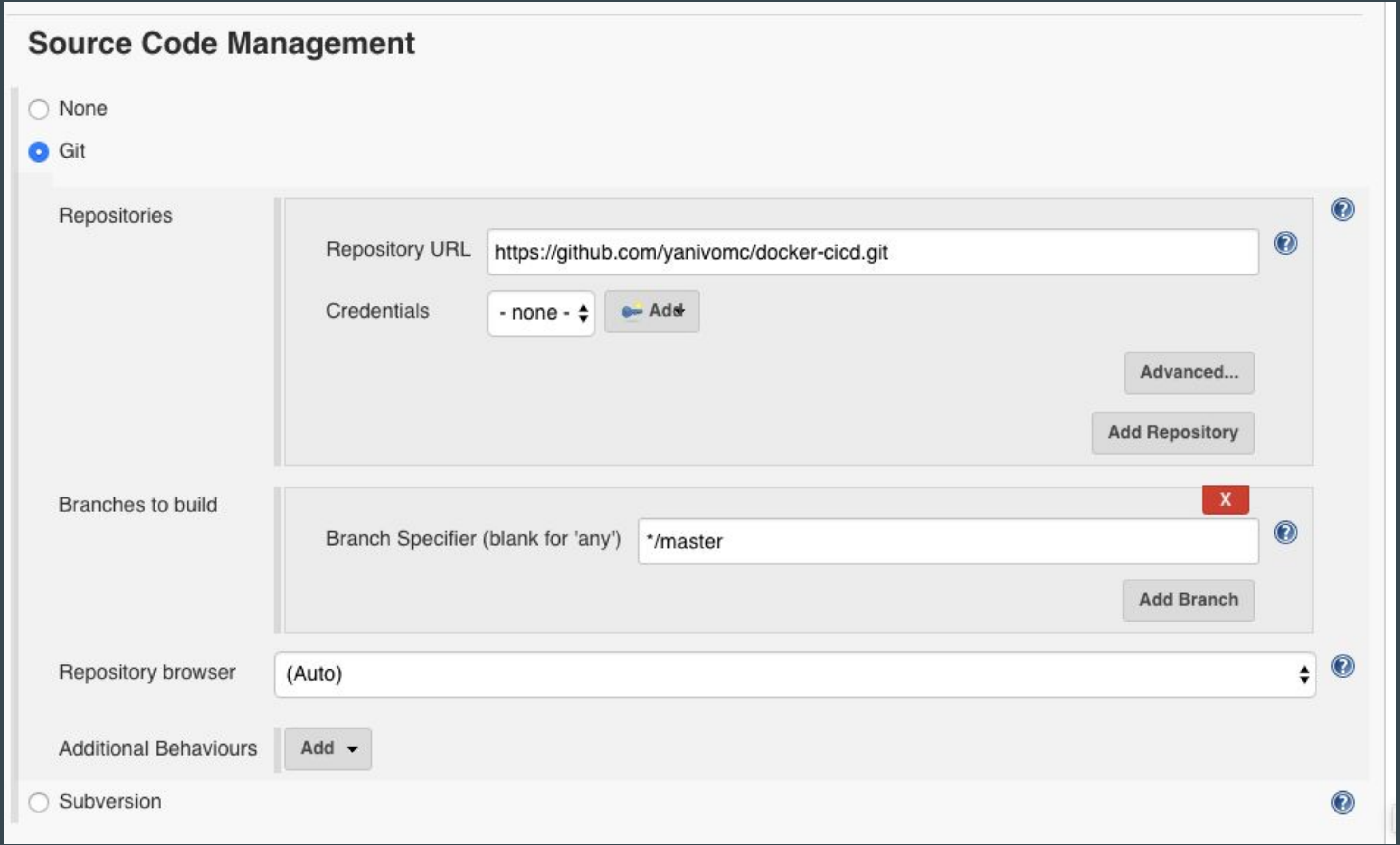
- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s)
- ☐ Provide Configuration files
- ☐ Abort the build if it's stuck
- ☐ Add timestamps to the Console Output
- ☑ Provide Node & npm bin/ folder to PATH

NodeJS Installation: NodeJS

Specify needed nodejs installation where npm installed packages will be provided to the PATH

npmrc file: -use system default-

BUILDING OUR FIRST APPLICATION



Source Code Management

None [✓ Git] Subversion

Repositories

Repository URL: https://github.com/yanivomc/docker-cicd.git

Credentials: - none -

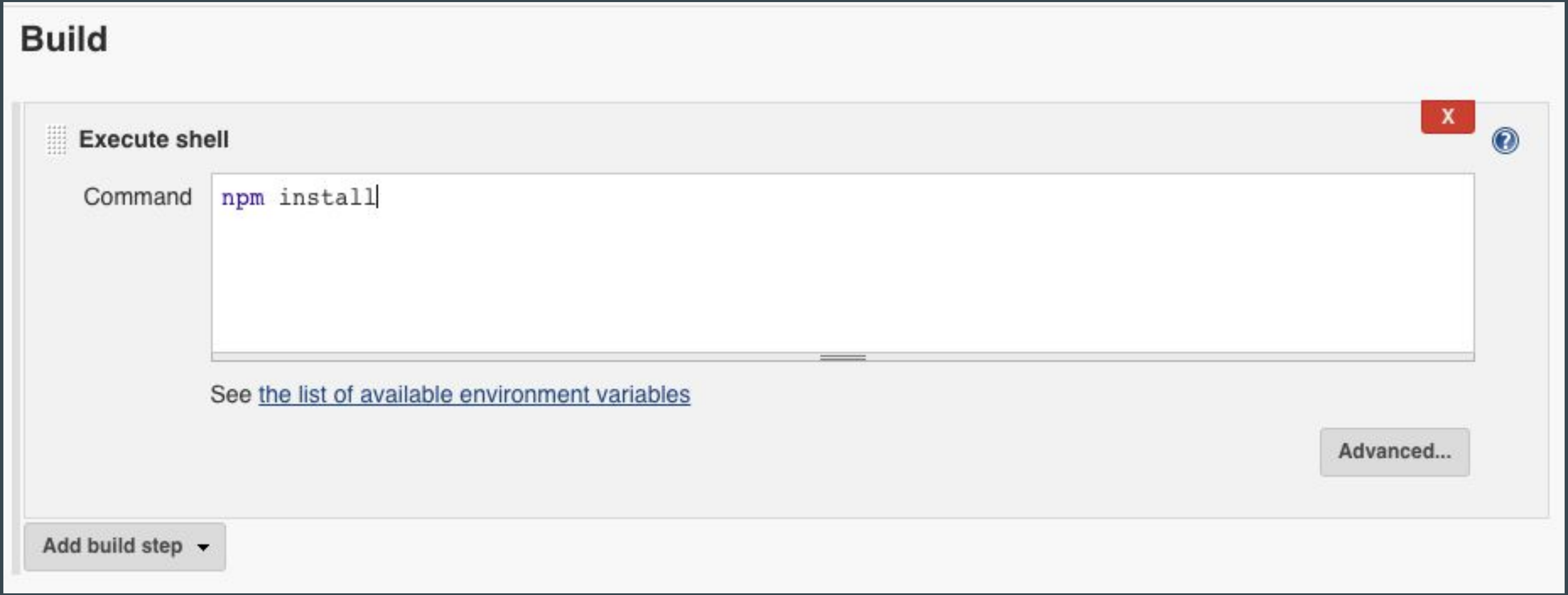
Branches to build

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Additional Behaviours

BUILDING OUR FIRST APPLICATION


Build

Execute shell

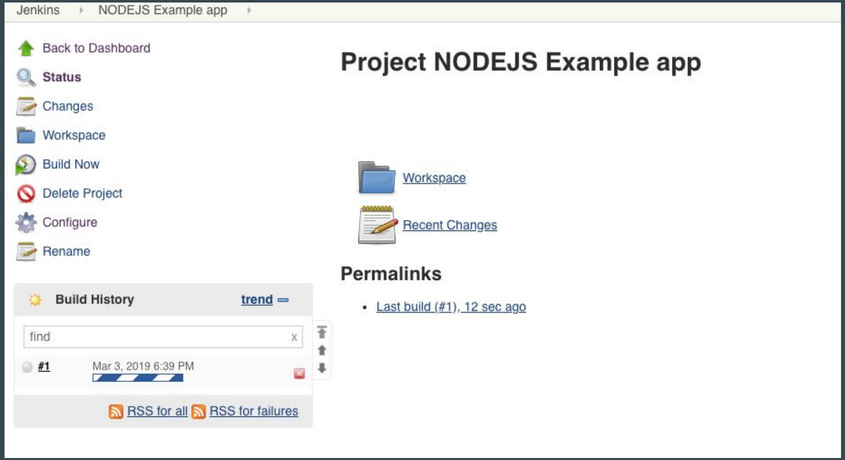
Command

```
npm install
```

See the list of available environment variables

BUILDING OUR FIRST APPLICATION

RUN THE JOB



Jenkins › NODEJS Example app ›

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- Rename

Project NODEJS Example app

- Workspace
- Recent Changes

Permalinks

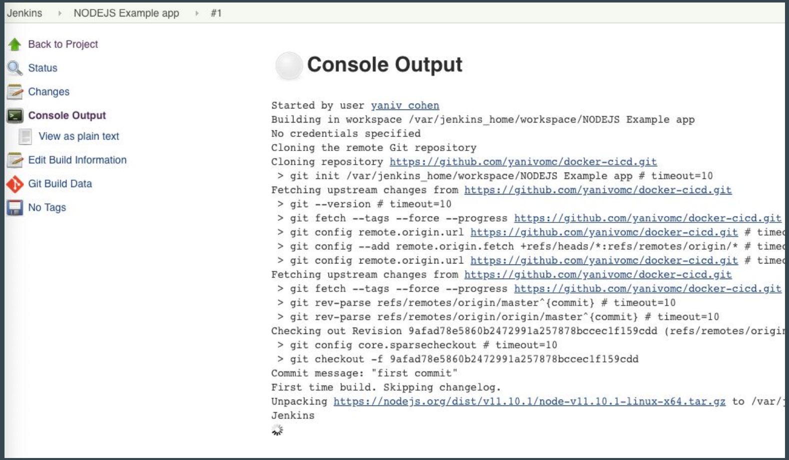
- Last build (#1), 12 sec ago

Build History trend

find

#1	Mar 3, 2019 6:39 PM

RSS for all RSS for failures



Jenkins › NODEJS Example app › #1

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- Edit Build Information
- Git Build Data
- No Tags

Console Output

```
Started by user yaniv cohen
Building in workspace /var/jenkins_home/workspace/NODEJS Example app
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/yanivomc/docker-cicd.git
 > git init /var/jenkins_home/workspace/NODEJS Example app # timeout=10
Fetching upstream changes from https://github.com/yanivomc/docker-cicd.git
 > git --version # timeout=10
 > git fetch --tags --force --progress https://github.com/yanivomc/docker-cicd.git
 > git config remote.origin.url https://github.com/yanivomc/docker-cicd.git # time
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # time
 > git config remote.origin.url https://github.com/yanivomc/docker-cicd.git # time
Fetching upstream changes from https://github.com/yanivomc/docker-cicd.git
 > git fetch --tags --force --progress https://github.com/yanivomc/docker-cicd.git
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 9afad78e5860b2472991a257878bccec1f159cdd (refs/remotes/origi
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 9afad78e5860b2472991a257878bccec1f159cdd
Commit message: "first commit"
First time build. Skipping changelog.
Unpacking https://nodejs.org/dist/v11.10.1/node-v11.10.1-linux-x64.tar.gz to /var/
Jenkins
```

View the files on the Jenkins FS

```
docker exec -ti jenkins-master bash
```

```
ls -ltrh /var/jenkins_home/workspace/NODEJS Example app
```

- MANUAL JOB - DONE -

- BUILDING NODE.JS APP - THE DOCKER STYLE

BUILDING OUR FIRST APPLICATION - DOCKER

CONFIGURING JENKINS

- **STEP I** Package our app in docker
  - Build a container that includes our NODE Application
  - Instead of creating a zip / tar / .gz... package. We will package our app and all it's binaries and dependencies inside a docker image.
  - This will make sure that our code will run the same - everytime and everywhere.

- **STEP II** Distribute the application using our newly created docker image
  - Build image
  - Push to docker repo

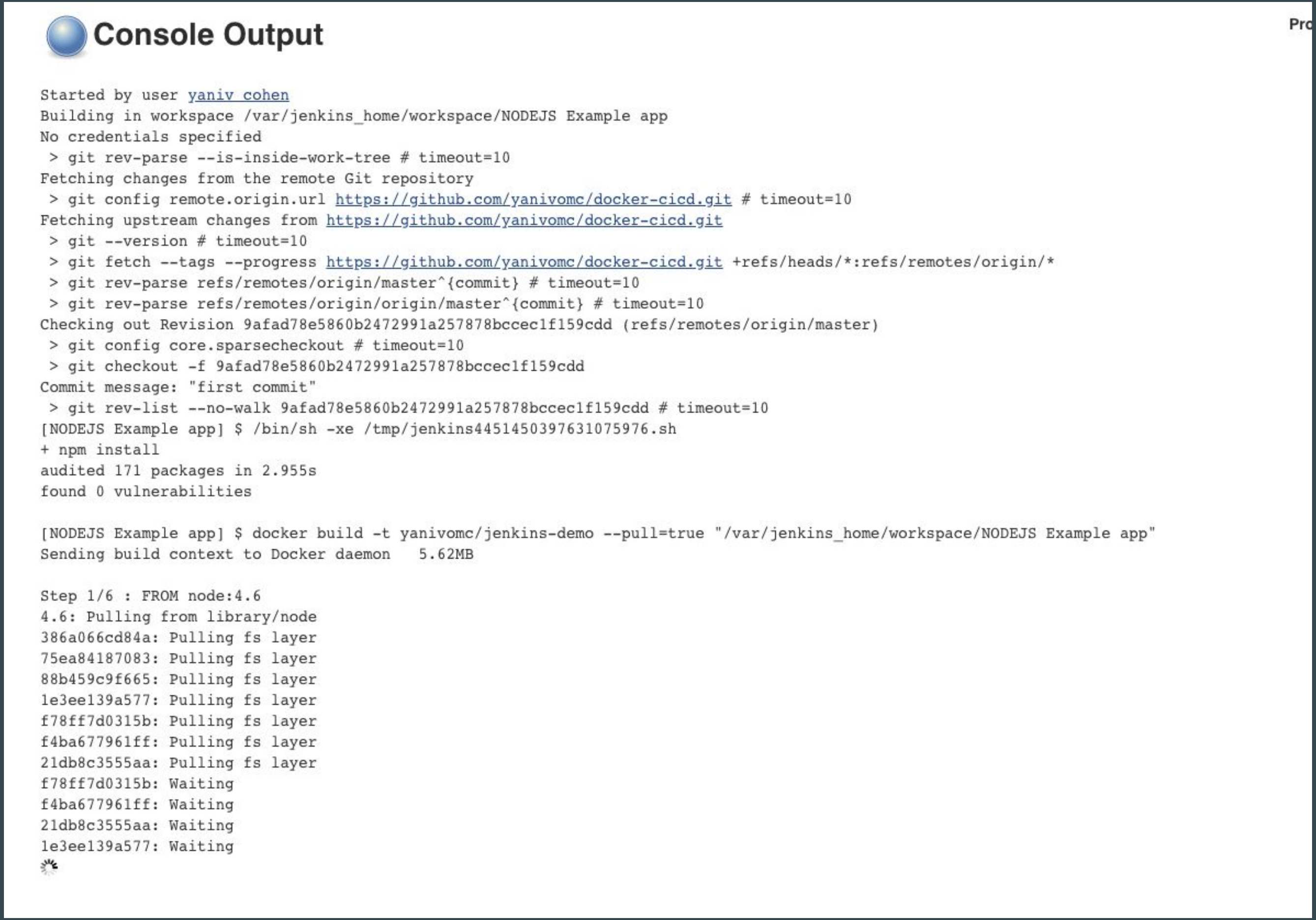
CONFIGURING JENKINS - LAB

- INSTALL CloudBees Docker Build and Publish Plugin
- OPEN/CREATE new DOCKER HUB REPO
- UPDATE and ADD build step Docker build and Publish
  - Update repo name
  - Add Registry credentials
    - Name and password only
- SAVE Job configuration
- Run DOCKER login inside your container:

```
docker exec -ti jenkins-master docker login
```

- Run the job and view the output

BUILDING OUR FIRST APPLICATION - DOCKER


Console Output

Pro

```
Started by user yaniv cohen
Building in workspace /var/jenkins_home/workspace/NODEJS Example app
No credentials specified
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/yanivomc/docker-cicd.git # timeout=10
Fetching upstream changes from https://github.com/yanivomc/docker-cicd.git
 > git --version # timeout=10
 > git fetch --tags --progress https://github.com/yanivomc/docker-cicd.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 9afad78e5860b2472991a257878bccec1f159cdd (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 9afad78e5860b2472991a257878bccec1f159cdd
Commit message: "first commit"
 > git rev-list --no-walk 9afad78e5860b2472991a257878bccec1f159cdd # timeout=10
[NODEJS Example app] $ /bin/sh -xe /tmp/jenkins4451450397631075976.sh
+ npm install
audited 171 packages in 2.955s
found 0 vulnerabilities

[NODEJS Example app] $ docker build -t yanivomc/jenkins-demo --pull=true "/var/jenkins_home/workspace/NODEJS Example app"
Sending build context to Docker daemon   5.62MB

Step 1/6 : FROM node:4.6
4.6: Pulling from library/node
386a066cd84a: Pulling fs layer
75ea84187083: Pulling fs layer
88b459c9f665: Pulling fs layer
1e3ee139a577: Pulling fs layer
f78ff7d0315b: Pulling fs layer
f4ba677961ff: Pulling fs layer
21db8c3555aa: Pulling fs layer
f78ff7d0315b: Waiting
f4ba677961ff: Waiting
21db8c3555aa: Waiting
1e3ee139a577: Waiting
```

BUILDING OUR FIRST APPLICATION - DOCKER

RUN OUR NEWLY CREATED APPLICATION

```
docker run -tid -p 3000:3000 --rm --name my-node-app reponame/image_name
```

```
BROWSE http://localhost:3000
```

- JENKINS AUTOMATION - IAC

PROBLEM

JENKINS ALLOWS US TO USE WEB UI TO INPUT ALL THE BUILD PARAMETERS

- Which leads to the following issues
  - No proper audit trail
  - No easy way to view history of changes
  - Segregations between groups
    - Users (developers) needs to contact jenkins admins to make a change?!?
    - Long iteration for changes
  - No Backup / Recover procedure
  - No automation basically - Sucks.

JENKINS AUTOMATION - IAC

SOLUTION

WRITE EVERYTHING AS A CODE...

- Version control it for history and Audit trail
- Fast rollback and updates
- Allows operations to give CONTROL to developers
  - Developers can and should bundle the JENKINS BUILD INSTRUCTIONS (code) with their own code repo (eg. JenkinsFile)
- Company needs and should embrace this Devops way to allow: developers to control their own builds and environments.

JENKINS AUTOMATION - IAC

Next section agenda:

- Jenkins DSL
    - Write code that creates and modifies **jenkins jobs** automatically
- Jenkins Pipeline (Jenkinsfile)
    - Bundle the **build parameters** within the project
    - Allow developers to change jenkins **BUILD** parameters

- JENKINS AUTOMATION - DSL

JENKINS AUTOMATION - IAC

Next section agenda:

- Jenkins DSL
  - Write code that creates and modifies **jenkins jobs** automatically
- Jenkins Pipeline (Jenkinsfile)
  - Bundle the **build parameters** within the project
  - Allow developers to change jenkins **BUILD** parameters

JENKINS AUTOMATION - IAC

Next section agenda:

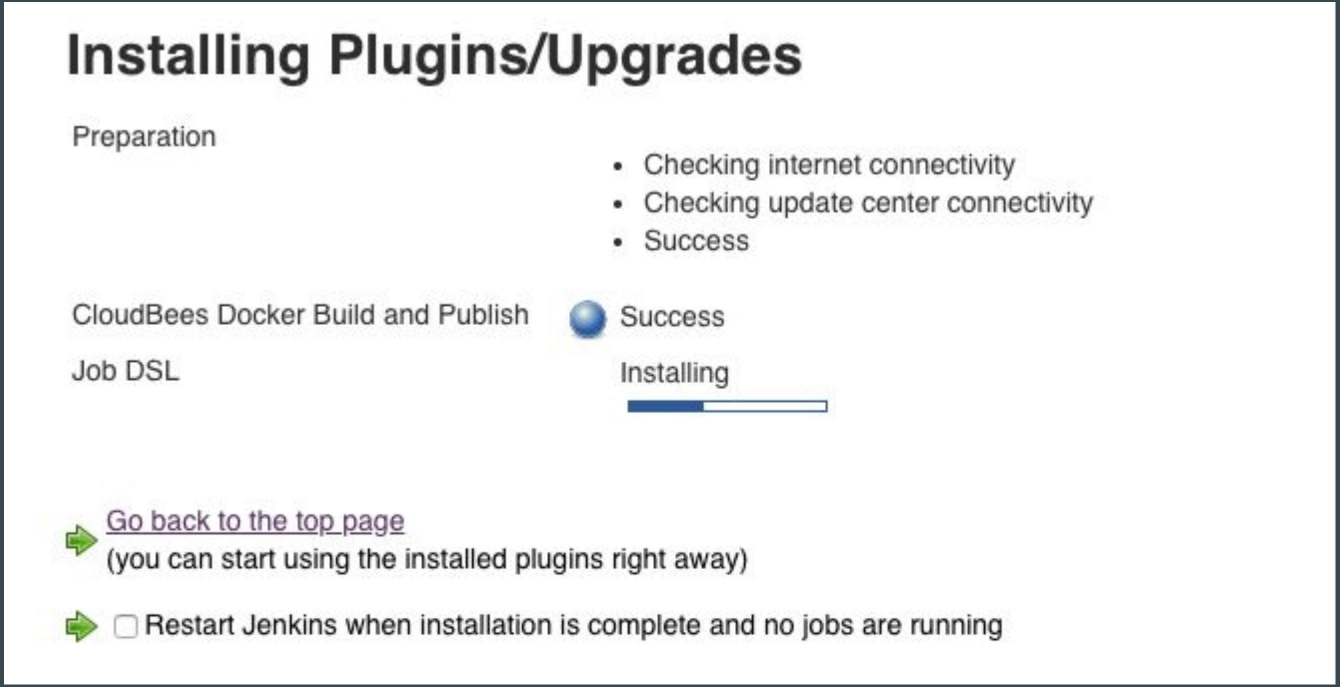
- Jenkins DSL
  - Write code that creates and modifies **jenkins jobs** automatically
- Jenkins Pipeline (Jenkinsfile)
  - Bundle the **build parameters** within the project
  - Allow developers to change jenkins **BUILD** parameters

DSL - Domain Specific Language

- The jenkins JOB DSL is a plugin that allows you to define jobs in a programmatic Form with minimal effort
- Using Jenkins DSL we can describe jobs using a Groovy based language
  - Think about Groovy as a powerful scripting language for the java platform
  - Similar to java BUT much more **simpler** because it's much more dynamic
- If we dont have many jobs , the UI might still be the easiest way to start until the number of jobs grow and becomes **difficult** to maintain.

- JENKINS AUTOMATION -
STEP 1 USING DSL TO CREATE A JOB

FIRST LET'S INSTALL THE JOB DSL PLUGIN



Installing Plugins/Upgrades

Preparation
- Checking internet connectivity
- Checking update center connectivity
- Success

CloudBees Docker Build and Publish ● Success

Job DSL Installing

Go back to the top page
(you can start using the installed plugins right away)

☐ Restart Jenkins when installation is complete and no jobs are running

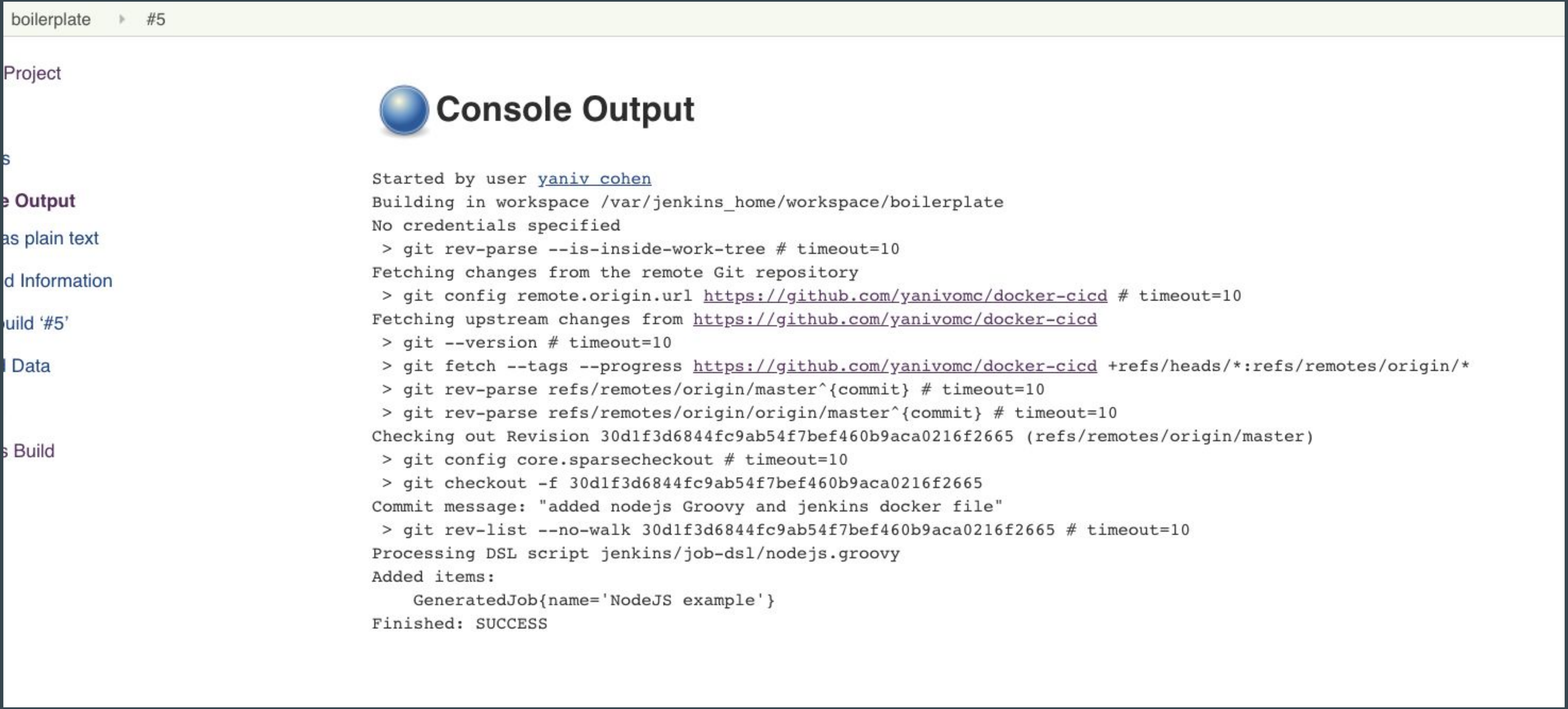
JENKINS AUTOMATION - NODEJS JOB DSL WAY

```
job('NodeJS example') { // Job NAME
    scm { // Configure Source control management
        git('git://github.com/yanivomc/docker-demo.git') {  node -> // is hudson.plugins.git.GitSCM
            node / gitConfigName('DSL User')
            node / gitConfigEmail('jenkins-dsl@domain.com)
        }
    }
    triggers { // Configure when to check for changes
        scm('H/5 * * * *')
    }
    wrappers {
        nodejs('nodejs') // this is the name of the NodeJS installation in
                         // Manage Jenkins -> Configure Tools -> NodeJS Installations -> Name
    }
    steps { // what steps to take
        shell("npm install")
    }
}
```

Follow Through : Running our NODEJS job using DSL

1. Create new freestyle job that will run our Groovy file
    a. Name: boilerplate
    b. Add git repo: https://github.com/yanivomc/docker-cicd
    c. On the Build step select: Process Job DSL
        i. Set DSL Scripts to point our folder where the groovy is located: “jenkins/job-dsl/nodejs.groovy”
        ii. Run The build and it will fail… with an error: “ERROR: script not yet approved for use” This happens since version 1.6 of Job DSL.
        iii. Jenkins would like you to first approve this script before it will be able to run it
            1. **Go to: Jenkins -> Manage -> In-process script approval and approve the script**
            2. **If you wish to auto allow all scripts uncheck: “**Enable script security for Job DSL scripts” under **Jenkins -> Manage -> Configure Global security**
        iv. ReRun the build

JENKINS AUTOMATION - NODEJS JOB DSL WAY



```
Started by user yaniv cohen
Building in workspace /var/jenkins_home/workspace/boilerplate
No credentials specified
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/yanivomc/docker-cicd # timeout=10
Fetching upstream changes from https://github.com/yanivomc/docker-cicd
 > git --version # timeout=10
 > git fetch --tags --progress https://github.com/yanivomc/docker-cicd +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 30d1f3d6844fc9ab54f7bef460b9aca0216f2665 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 30d1f3d6844fc9ab54f7bef460b9aca0216f2665
Commit message: "added nodejs Groovy and jenkins docker file"
 > git rev-list --no-walk 30d1f3d6844fc9ab54f7bef460b9aca0216f2665 # timeout=10
Processing DSL script jenkins/job-dsl/nodejs.groovy
Added items:
    GeneratedJob{name='NodeJS example'}
Finished: SUCCESS
```

- JENKINS AUTOMATION - STEP 2 DOCKER BUILD

JENKINS AUTOMATION - NODEJS JOB DSL WAY - DOCKER

```
job('NodeJS Docker example') {
    scm {
        git('git://github.com/wardviaene/docker-demo.git') {  node -> // is hudson.plugins.git.GitSCM
            node / gitConfigName('DSL User')
            node / gitConfigEmail('jenkins-dsl@newtech.academy')
        }
    }
    triggers {
        scm('H/5 * * * *')
    }
    wrappers {
        nodejs('nodejs') // this is the name of the NodeJS installation in
                         // Manage Jenkins -> Configure Tools -> NodeJS Installations -> Name
    }
    steps {
        dockerBuildAndPublish {
            repositoryName(ac/dc)
            tag('${GIT_REVISION,length=9}')
            registryCredentials('dockerhub')
            forcePull(false)
            forceTag(false)
            createFingerprints(false)
            skipDecorate()
        }
    }
}
```

JENKINS AUTOMATION - NODEJS JOB DSL WAY - DOCKER

LAB: ADDING SECOND STEP TO BUILD THE DOCKER

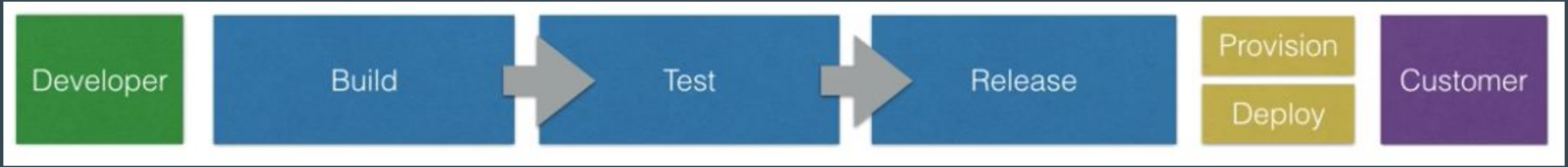
1. Update our job configuration in UI to run the new groovy file
   a. Groovy Location: https://github.com/yanivomc/docker-cicd/blob/master/jenkins/job-dsl/nodejs-docker.groovy
   b. Info about the plugin options: https://jenkinsci.github.io/job-dsl-plugin/#method/javaposse.jobdsl.dsl.helpers.step.StepContext.dockerBuildAndPublish
   c. Errors expected
      i. Try to resolve them

- JENKINS AUTOMATION -
JENKINS PIPELINES

JENKINS AUTOMATION - PIPELINES

Jenkins PIPELINES allow us to write the Jenkins Build steps in code

1. Build steps allow us to write build (compile) , test , deploy in code
2. Code means we can put this code in our VS / SCM
3. With the above create we can create a full CI/CD Flow



JENKINS PIPELINES

- Is a **specific job type** that can be created using UI or DSL
- We can use to write the pipeline in **Jenkins DSL** (Declarative pipeline) or in **Groovy** (Scripted pipeline).

** Under the hood jenkins runs JENKINS DSL in JavaVM as Groovy

JENKINS AUTOMATION - PIPELINES

```
node { // On which node this pipeline will run
   def commit_id // Create variables. Commit_id is exposed automatically by jenkins
   stage('Preparation') { // Job STAGE’E
     checkout scm // Check out this repo
     sh "git rev-parse --short HEAD > .git/commit-id"  // Get the latest commit-id
     commit_id = readFile('.git/commit-id').trim() // store the commit-id in the defined variable
   }
   stage('test') { // Stage TEST
     nodejs(nodeJSInstallationName: 'nodejs') {
       sh 'npm install --only=dev' // RUN npm install dev
       sh 'npm test' // RUN npm tests
     }
   }
   stage('docker build/push') { // Stage buif oush
     docker.withRegistry('https://index.docker.io/v1/', 'dockerhub') { // used docker hub api and ‘docker hub’
                                                                          credintials
       def app = docker.build("repo/imagename:${commit_id}", '.').push()
     }
   }
}
```

Follow Through : Running our NODEJS job using PIPELINE

1. Create new pipeline job that will run our JenkinsFile
   a. Name: boilerplate-pipeline
   b. On Pipline Add git repo: https://github.com/yanivomc/docker-cicd
   c. Script Path: misc/Jenkinsfile
   d. Run the build

- JENKINS AUTOMATION -
JENKINS PIPELINE TESTS WITH DOCKER

Follow Through : LETS REVIEW

1. In this follow through we will see how we can run everything within a docker container
   a. Meaning that we will run all steps inside a docker container instead of using external plugins for NodeJS For example or different version per build etc.
2. Lets review the file - https://github.com/yanivomc/docker-cicd/blob/master/misc/Jenkinsfile.v2

- JENKINS AUTOMATION -
LAB: RUN THE PIPELINE

- JENKINS INTEGRATION -
GITHUB / BITBUCKET INTEGRATION

JENKINS INTEGRATION - GITHUB / BITBUCKET

UP UNTIL NOW WE'VE MANUALLY ADDED GIT REPO'S

1. In cases where we have a lot of repo's,
   We dont wish to add every single repo manually right ?
2. Using Jenkins Integration plugins to Github and Bitbucket we can allow jenkins to auto detect new repo's
3. Once a developer create a new repo for his new (micro) service,
   Add a Jenkinsfile, the project will automatically be built in Jenkins!

PLUGINS

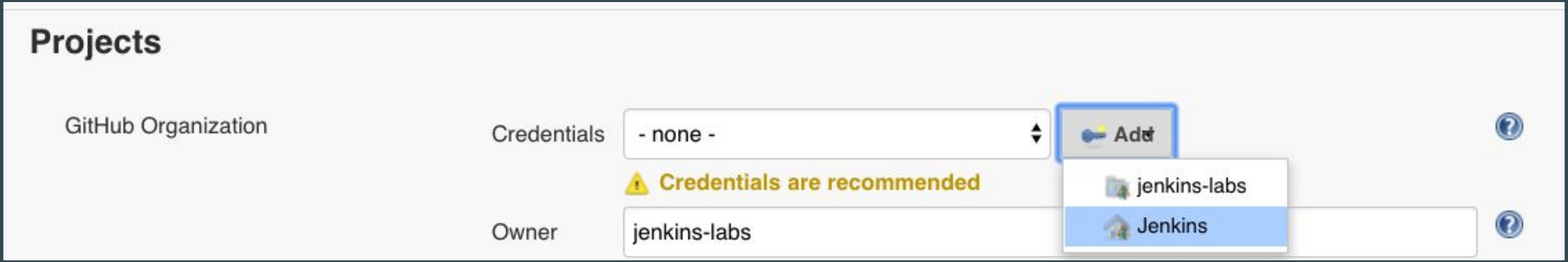
1. GITHUB: GitHub Branch Source Plugin
2. BITBUCKET: Bitbucket Branch Source Plugin

- JENKINS INTEGRATION -
LAB: GITHUB INTEGRATION

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO

1. Install the github plugin
2. In your GITHUB account create a new personal token to be used by Jenkins
   a. https://github.com/settings/tokens/new
3. Create new job type "GitHub Organization"
   a. Type name and on Projects github credentials add new global credenials



Projects

GitHub Organization

Credentials: - none -

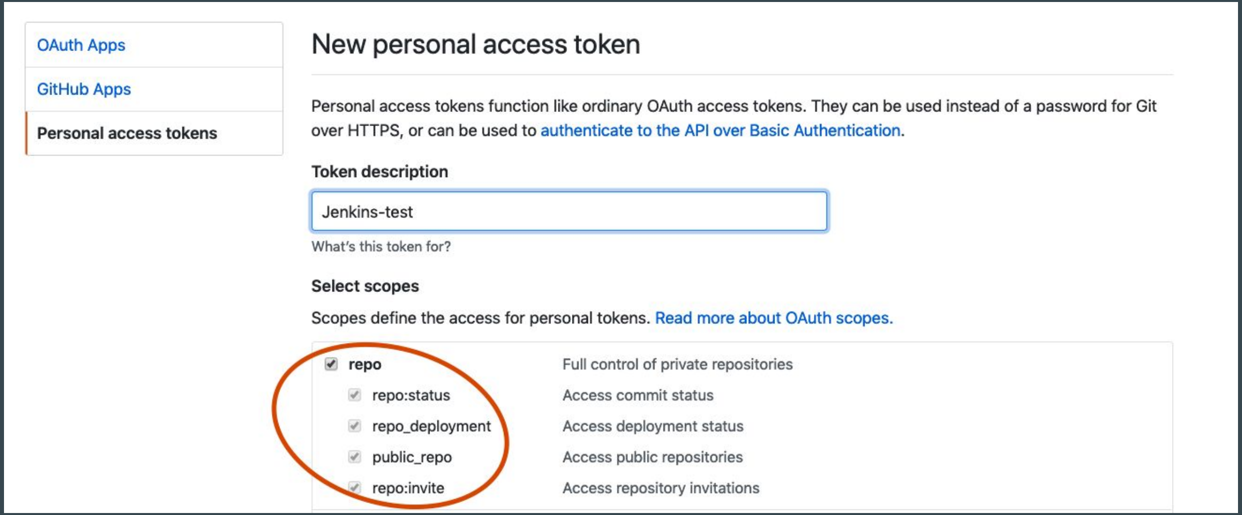
Credentials are recommended

Owner: jenkins-labs

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO

1. Install the github plugin
2. In your GITHUB account create a new personal token to be used by Jenkins
   a. https://github.com/settings/tokens/new



OAuth Apps
GitHub Apps
Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Token description

Jenkins-test

What's this token for?

Select scopes

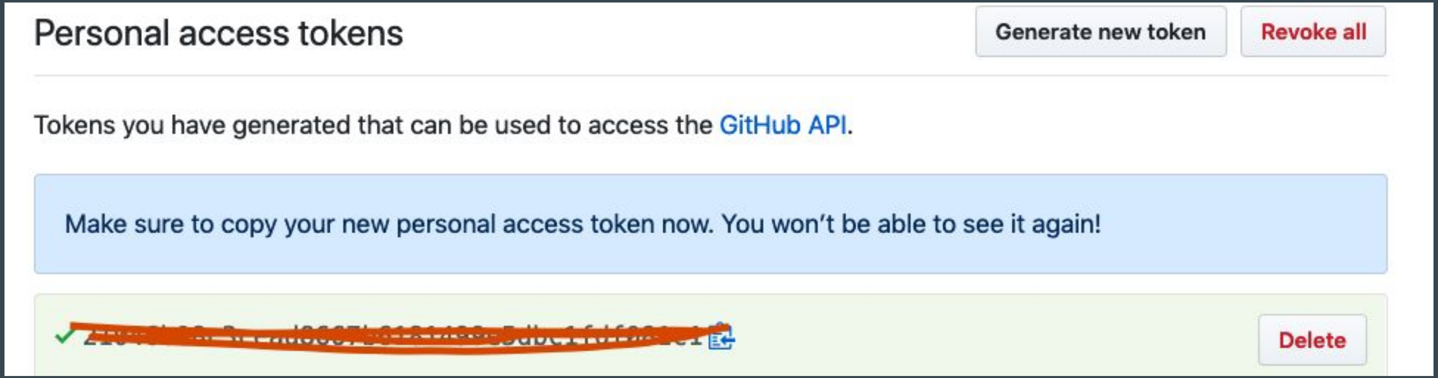
Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ **repo**	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO

1. Install the github plugin
2. In your GITHUB account create a new personal token to be used by Jenkins
   a. https://github.com/settings/tokens/new



Tokens you have generated that can be used to access the GitHub API.

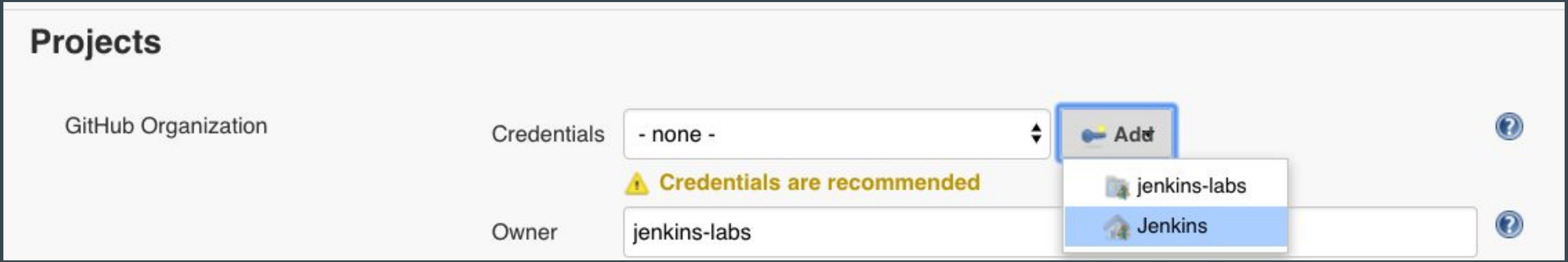
Make sure to copy your new personal access token now. You won't be able to see it again!

✓ [redacted]

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO

1. Install the github plugin
2. In your GITHUB account create a new personal token to be used by Jenkins
   a. https://github.com/settings/tokens/new
3. Create new job type "GitHub Organization"
   a. Type name and on Projects github credentials add new global credenials



Projects

GitHub Organization

Credentials: - none -

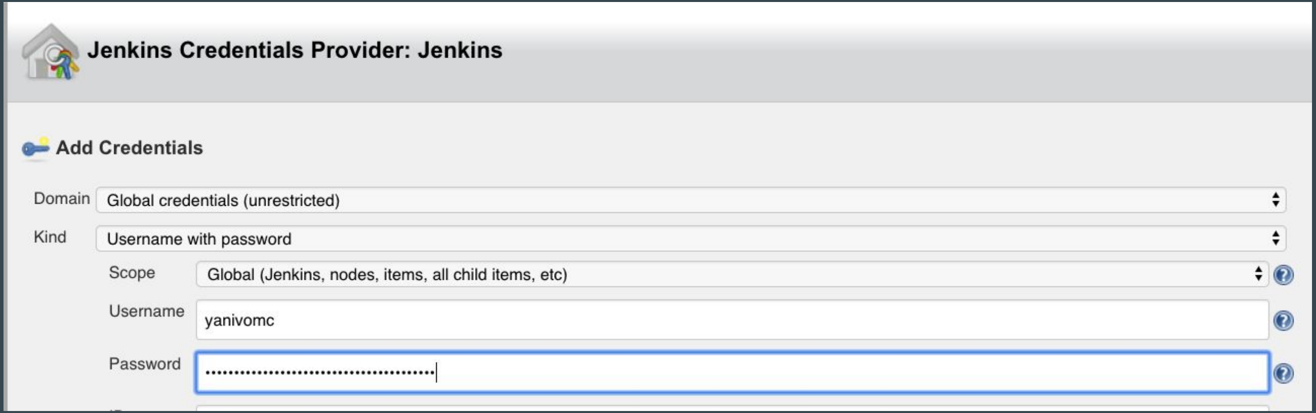
⚠ Credentials are recommended

Owner: jenkins-labs

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO

1. Install the github plugin
2. In your GITHUB account create a new personal token to be used by Jenkins
   a. https://github.com/settings/tokens/new
3. Create new job type "GitHub Organization"
   a. Type name and on Projects github credentials add new global credentials



Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

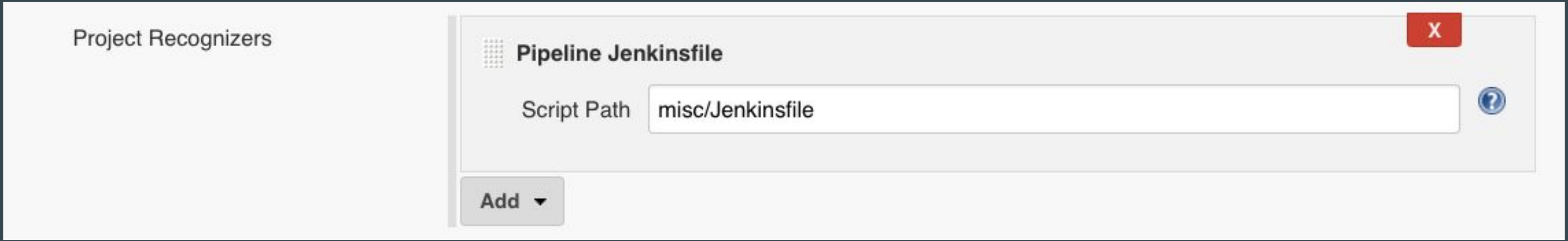
Username: yanivomc

Password: ••

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO

1. Install the github plugin
2. In your GITHUB account create a new personal token to be used by Jenkins
   a. https://github.com/settings/tokens/new
3. Create new job type “GitHub Organization”
   a. Type name and on Projects github credentials add new global credentials
   b. Update the Owner name to your username in github
   c. Point project recognizers to the location of our Jenkinsfile in our repo



Project Recognizers

Pipeline Jenkinsfile

Script Path: misc/Jenkinsfile

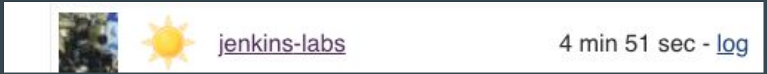
- JENKINS INTEGRATION - RUN

JENKINS INTEGRATION - GITHUB / BITBUCKET

DEMO EXPLAINED

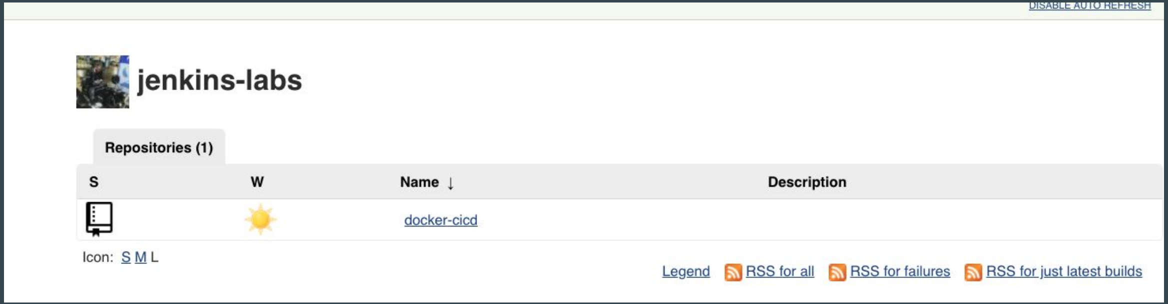
1. Jenkins will scan all of our repos (or the repos we asked to be scanned)
2. Once a new or already exists repo identified, jenkins will look for jenkinsfile in it
3. Once the jenkinsfile was found - Jenkins will automatically build a new job and run it

4. A new view will be added



jenkins-labs 4 min 51 sec - log

5. Once we click on the new view we will see new jobs per repos pipeline



DISABLE AUTO REFRESH

jenkins-labs

Repositories (1)

S	W	Name ↓	Description
	☀	docker-cicd	

Icon: S M L

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

- MAKING A CHANGE -

Thank you and see you soon

●●●

Provided by DevopShift for JB & Presented by DevopShift CEO Yaniv Cohen