****

DevSecOps User Manual

**Index**:

[Pre- Requisites:](#_jj1klkoy3qox) **1**

[What is DevSecops?](#_ajnj4649ms7z) **1**

[What are we using in DevSecOps?](#_af2o6no48kfo) **1**

[How to Setup Credentials?](#_keyipddlqdxh) **12**

[Creating Gitlab Access Token](#_4ksqps6sfyfm) 12

[Setting credentials in Jenkins before running the pipelines](#_bd8r85s93cz7) 15

[**How to use DevSecOps Pipeline?**](#_xj7gtparevk) **17**

[How to view/download pipeline results?](#_x5hbuuraz27m) **22**

## **Pre- Requisites:**

* If you are using python language then make sure requirements.txt file available in the root folder of the project.
* Before Running the DevSecOps pipeline kindly [setup the credentials](#_keyipddlqdxh) to access the code repositories from Gitlab

## **What is DevSecops?**

DevSecOps (**Dev**elopment, **Sec**urity and **Op**erations) —automates the integration of security at every phase of the software development lifecycle, from initial design through integration, testing, deployment, and software delivery by leveraging applicable tools.

## **What are we using in DevSecOps?**

We are using different stages to perform the security scan that includes Credential Checks, SCA(Software Composition Analysis), SAST(Static Application Security Testing), DAST(Dynamic Application Security Testing, IAC(Infrastructure as Code), and Vulnerability Management.

**Credential Checks:** Git Credentials checks include some tools that perform credentials checks on the commits to find the sensitive information and other security misconfiguration.

* We are using trufflehog, talisman, and git-secrets for now to perform the credentials checks.

1. **Talisman:** Talisman is a tool that installs a hook to your repository to ensure that potential secrets or sensitive information do not leave the developer's workstation.

It validates the outgoing changeset for things that look suspicious - such as potential SSH keys, authorization tokens, private keys etc.

**Download link:** <https://github.com/thoughtworks/talisman>  
**Setup:   
1.** Talisman can be installed and used either as a git hook as a global git hook template and a CLI utility (for git repo scanning) or as a git hook into a single git repository.  
**2.** We recommend installing Talisman as a pre-commit git hook template, as that will cause Talisman to be present, not only in your existing git repositories, but also in any new repository that you 'init' or 'clone'.  
**3.** Run the following command on your terminal, to download and install the binary at $HOME/.talisman/bin

**As a pre-commit hook:** *curl --silent https://raw.githubusercontent.com/thoughtworks/talisman/master/global\_install\_scripts/install.bash > /tmp/install\_talisman.bash && /bin/bash /tmp/install\_talisman.bash*  
**OR**  
**As a pre-push hook:** *curl --silent https://raw.githubusercontent.com/thoughtworks/talisman/master/global\_install\_scripts/install.bash > /tmp/install\_talisman.bash && /bin/bash /tmp/install\_talisman.bash pre-push*  
**4.** If you do not have TALISMAN\_HOME set up in your $PATH, you will be asked for an appropriate place to set it up. Choose the option number where you set the profile source on your machine.  
**5.** Remember to execute the source on the path file or restart your terminal. If you choose to set the $PATH later, please export TALISMAN\_HOME=$HOME/.talisman/bin to the path.  
**6.** Choose a base directory where Talisman should scan for all git repositories, and set up a git hook (pre-commit or pre-push, as chosen in step 1) as a symlink.

1. **TruffleHog:** It searches through git repositories for secrets, digging deep into commit history and branches. This is effective at finding secrets accidentally committed.   
   **Download:** pip3 install trufflehog  
   **Usage:** trufflehog {git\_repo\_path\_or\_url}  
   The output consists of many messages like this one:  
   
2. **Git-Secret:** git-secret is a bash tool which stores private data inside a git repo. git-secret encrypts files with permitted users' public keys, allowing users you trust to access encrypted data using pgp and their secret keys. When someone's permission is revoked, secrets do not need to be changed with git-secret - just remove their key from the keychain using git secret kill person their@email.com, re-encrypt the files, and they won't be able to decrypt secrets anymore.   
   **Download link:** <https://github.com/sobolevn/git-secret.git>  
   **Setup:   
   1.** git-secret supports brew, just type: brew install git-secret.  
   **2.** It also supports apt and yum. You can also use make if you want to.  
   A preview of some functionality of git-secret

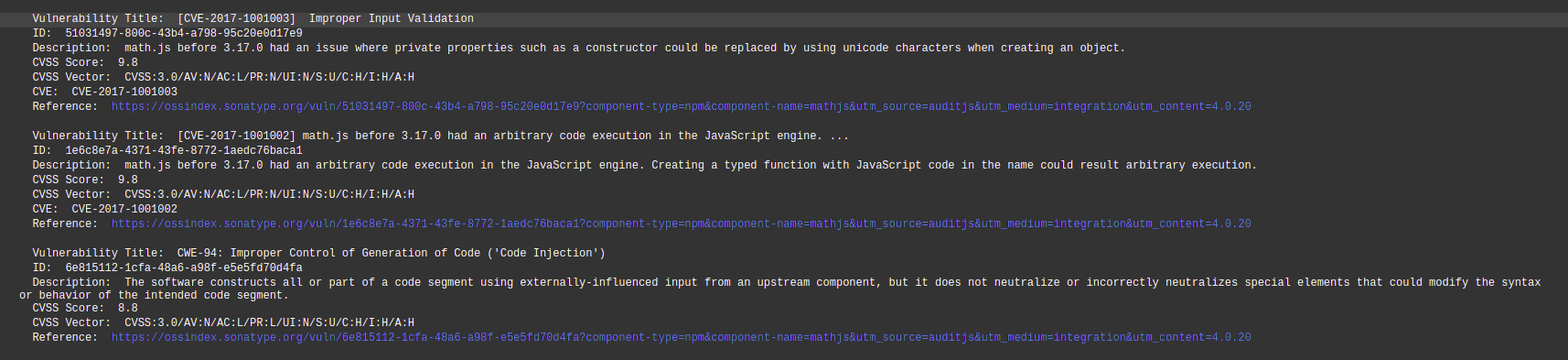
**SCA(Software Composition Analysis):** Software Composition Analysis (SCA) is the process of automating visibility into the use of open source software (OSS) for the purpose of risk management, security, and license compliance.

**Static code analysis** is a method of debugging by examining **source code** before a program is run. It's done by analyzing a set of **code** against a set (or multiple sets) of coding rules.

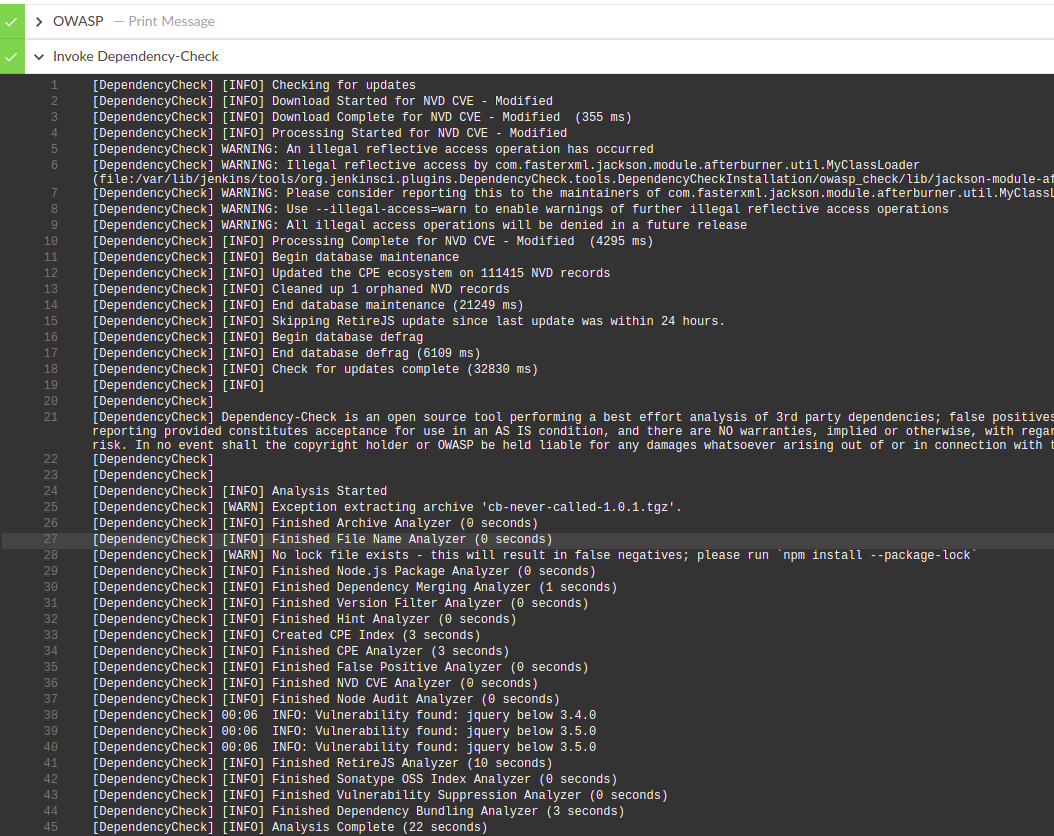
* Tools like auditjs, retirejs, and OWASP dependency checker will perform the scan and will provide the available security misconfiguration and security issues.

1. **Auditjs:** AuditJS functions by traversing your node\_modules folder in your project, so it will pick up the dependencies that are physically installed. Execute from inside a node project (above the node\_modules directory) to audit the dependencies. This will audit not only the direct dependencies of the project, but all **transitive** dependencies. To identify transitive dependencies they must all be installed for the project under audit.If a vulnerability is found to be affecting an installed library the package header will be highlighted in red and information about the pertinent vulnerability.

**Download:** npm install -g auditjs

  
E.g: Auditjs scans the code repository and find out the known vulnerabilities and outdated package version(As shown in adobe picture)

1. **Retirejs:** The goal of Retire.js is to help you detect the use of JS-library versions with known vulnerabilities.  
   **Download:** npm install -g retire
2. **OWASP Dependency Checker:** Dependency-Check is a Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within a project’s dependencies. It does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries.  
   **Download link:** <https://github.com/jeremylong/DependencyCheck>

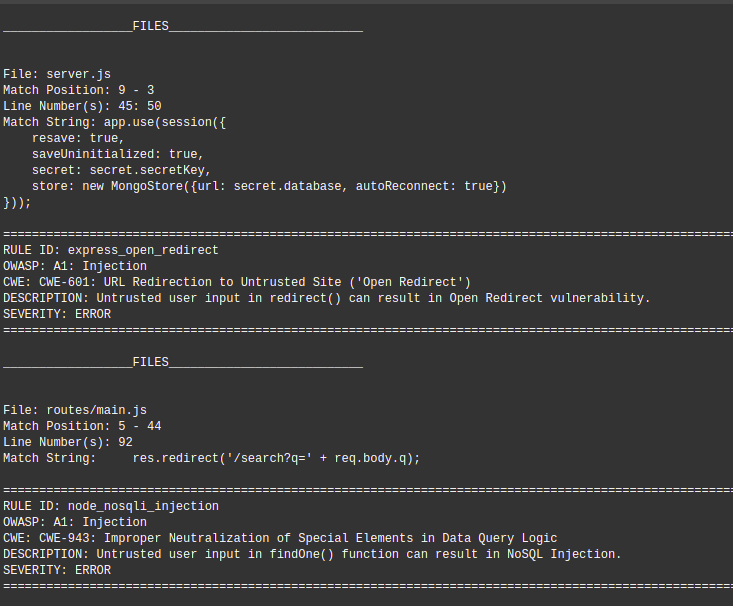


E.g: Owasp dependency checker attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies.

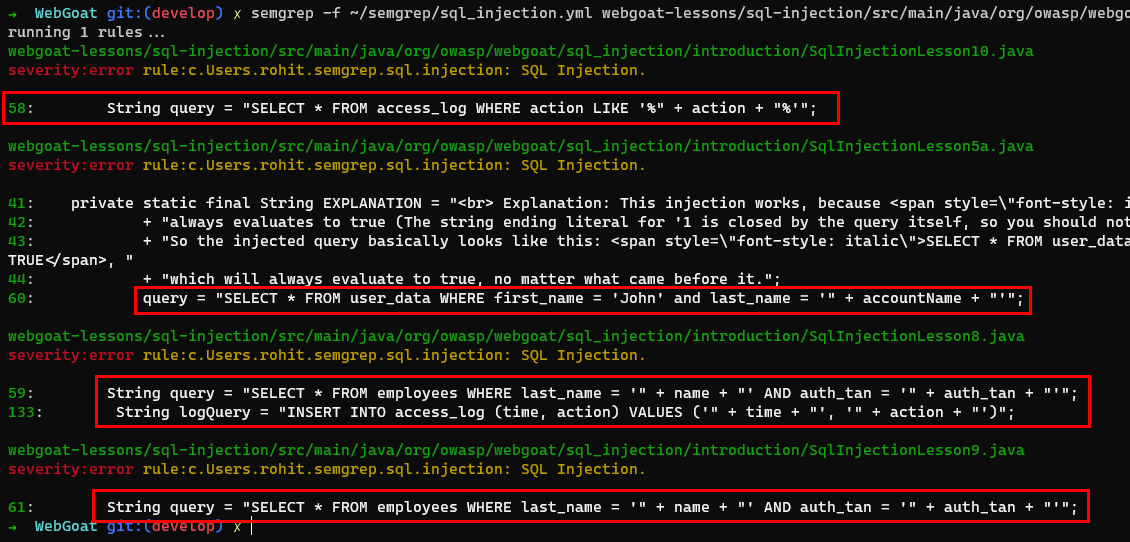
**SAST(Static Application Security Testing): SAST** is a set of technologies designed to analyze application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities. SAST solutions analyze an application from the “inside out” in a non running state.

* Static application security testing products scan the source code to identify susceptibilities, provide reports, and even develop code fixes for some of those vulnerabilities.
* The tools used to perform SAST testing are as follows.

1. **Njsscan**: **njsscan** is a static application testing (SAST) tool that can find insecure code patterns in your node.js applications using simple pattern matcher from [libsast](https://github.com/ajinabraham/libsast) and syntax-aware semantic code pattern search tool [semgrep](https://github.com/returntocorp/semgrep).   
   **Download link:** <https://github.com/ajinabraham/njsscan> **Alternative:** pip3 install njsscan  
   An example output of njsscan



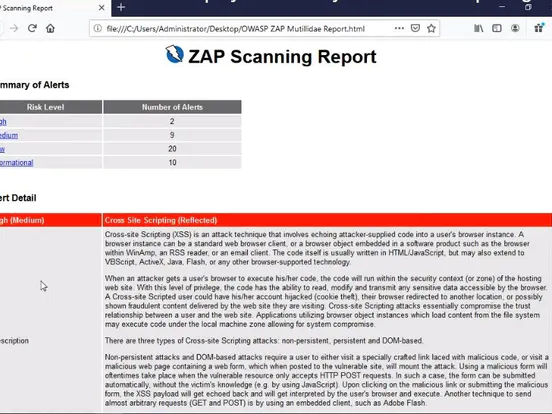
1. **Bandit**: Bandit is a tool designed to find common security issues in Python code. To do this Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report.  
   **Download:** pip3 install bandit
2. **Snyk:** Enabling developers to easily find and automatically fix open source vulnerabilities. **Find** known vulnerabilities by running snyk test on a project either as a one off or as part of your CI process. **Fix** vulnerabilities using snyk wizard and snyk protect. **Alert** snyk monitor records the state of dependencies and any vulnerabilities on snyk.io so you can be alerted when new vulnerabilities or updates/patches are disclosed that affect your repositories.  
   **Download:** npm install -g snyk
3. **Semgrep:** Semgrep is a fast, open-source, static analysis tool that excels at expressing code standards — without complicated queries — and surfacing bugs early at editor, commit, and CI time. Precise rules look like the code you’re searching; no more traversing abstract syntax trees or wrestling with Regular Expressions.  
   **Download link:** <https://github.com/returntocorp/semgrep.git>



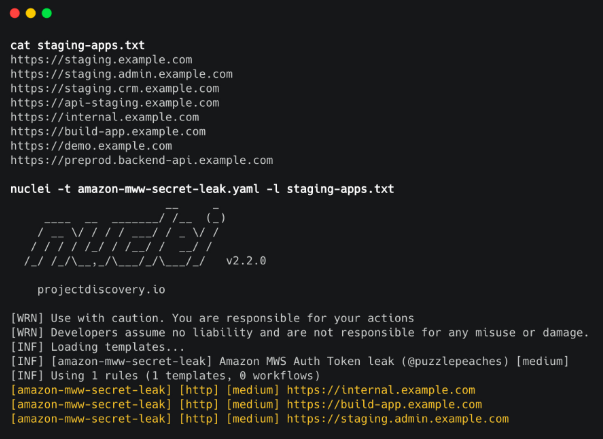
**DAST(Dynamic Application Security Testing):** Dynamic Application Security Testing (DAST) is a security solution that can help in finding vulnerabilities in web applications. It is performed in order to identify potential security vulnerabilities in the web application and architectural weaknesses.

* The post-exploitation stage is focussed on data-exfiltration and baoor implanting to have persistent access.
* Enumerating the compromised instance for any credentials and performing lateral movement for further exploitations are parts of DAST.
* The tools used to perform DAST testing are as follows.

1. **OWASP ZAP Scanner:** OWASP ZAP is an open-source web application security scanner. It is intended to be used by both those new to application security as well as professional penetration testers. Some of the built in features include: Intercepting proxy server, Traditional and AJAX Web crawlers, Automated scanner, Passive scanner, Forced browsing, Fuzzer, WebSocket support, Scripting languages, and Plug-n-Hack support.   
   **Download link:** <https://www.zaproxy.org/download/>

****

1. **Nuclei:** Nuclei is used to send requests across targets based on a template leading to zero false positives and providing fast scanning on a large number of hosts. Nuclei offers scanning for a variety of protocols including TCP, DNS, HTTP, File, etc. With powerful and flexible templating, all kinds of security checks can be modelled with Nuclei.   
   **Download link:** <https://github.com/projectdiscovery/nuclei.git>

****

1. **Sonarqube:** SonarQube is an open-source platform developed for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities in 20+ programming languages.   
   **Download link:** <https://www.sonarqube.org/downloads/>
2. **Burp Suite Pro:** “Burp,” as it is commonly known, is a proxy-based tool used to evaluate the security of web-based applications and do hands-on testing. You can intercept requests, run crawlers, modify requests, run automated scans to check for vulnerabilities, and much more.   
   **Download link:** <https://portswigger.net/burp/pro>

**IAC(Infrastructure as Code): IaC** is the managing and provisioning of infrastructure through code instead of through manual processes. With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations. It also ensures that you provision the same environment every time.

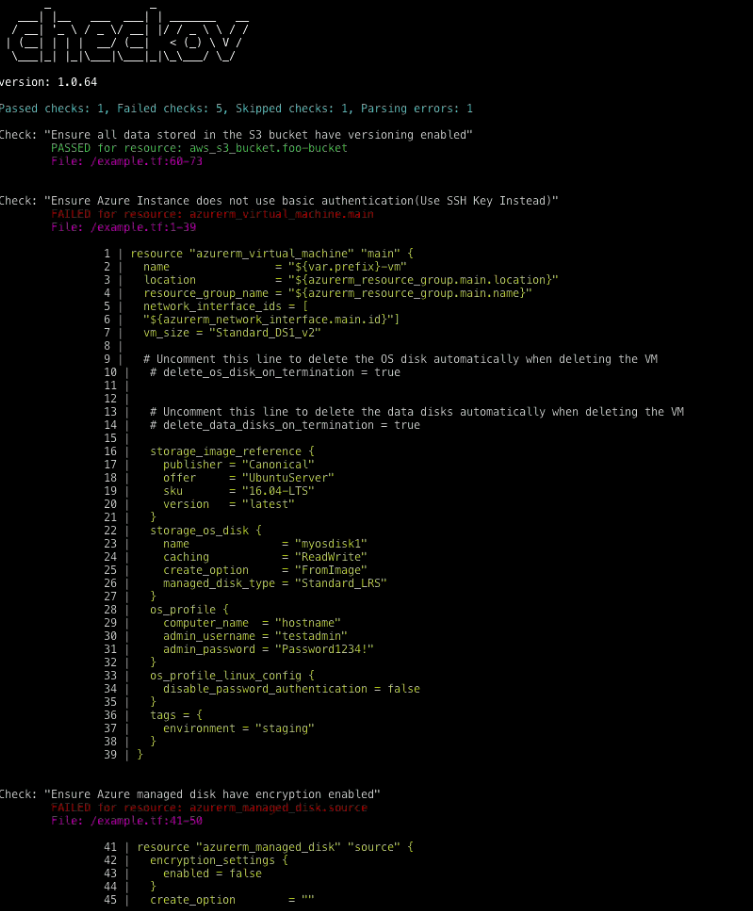
* The tools used for IaC testing are as follows.

1. **Tfsec:** tfsec uses static analysis of your terraform templates to spot potential security issues.   
   **Download link:** <https://github.com/tfsec/tfsec.git>



1. **Checkov:** Checkov is a static code analysis tool for infrastructure-as-code. It scans cloud infrastructure provisioning using Terraform, Terraform plan, Cloudformation, Kubernetes, Serverless or ARM Templates and detects security and compliance misconfigurations.

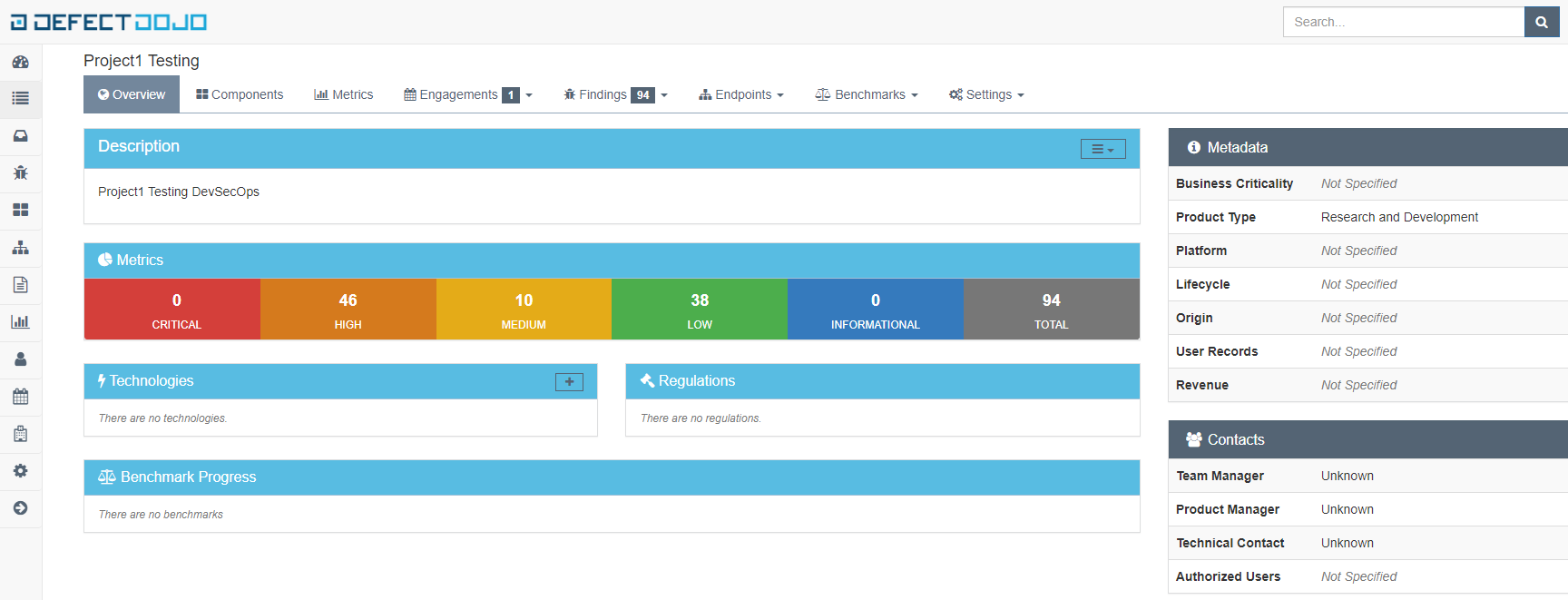
**Download link:** <https://github.com/bridgecrewio/checkov>



**Vulnerability Management:** For security purposes, It is important to integrate vulnerability management before the application into production. A DevSecOps model involves testing for vulnerabilities across the development and integration pipeline and again once the code is sent to production.

* The tools used for Vulnerability Management are as follows.

1. **DefectDojo:** DefectDojo is a security program and vulnerability management tool. DefectDojo allows you to manage your application security program, maintain product and application information, schedule scans, triage vulnerabilities and push findings into defect trackers. Consolidate your findings into one source of truth with DefectDojo. **Download link:** <https://github.com/DefectDojo/django-DefectDojo>



1. **Jira:** Jira is used for bug tracking and project management. Jira has evolved into a powerful tool for work management for all kinds of use cases, from requirements and test case management to agile software development. It is incredibly flexible and can be customized to work with the team’s unique workflow, meaning teams of all kinds can enjoy increased productivity and visibility.   
   **Download link:** <https://www.atlassian.com/software/jira/download/data-center>

## **How to Setup Credentials?**

Follow the below mentioned steps to create a access token on Gitlab and setup credentials ID in Jenkins to access the Gitlab code repositories.

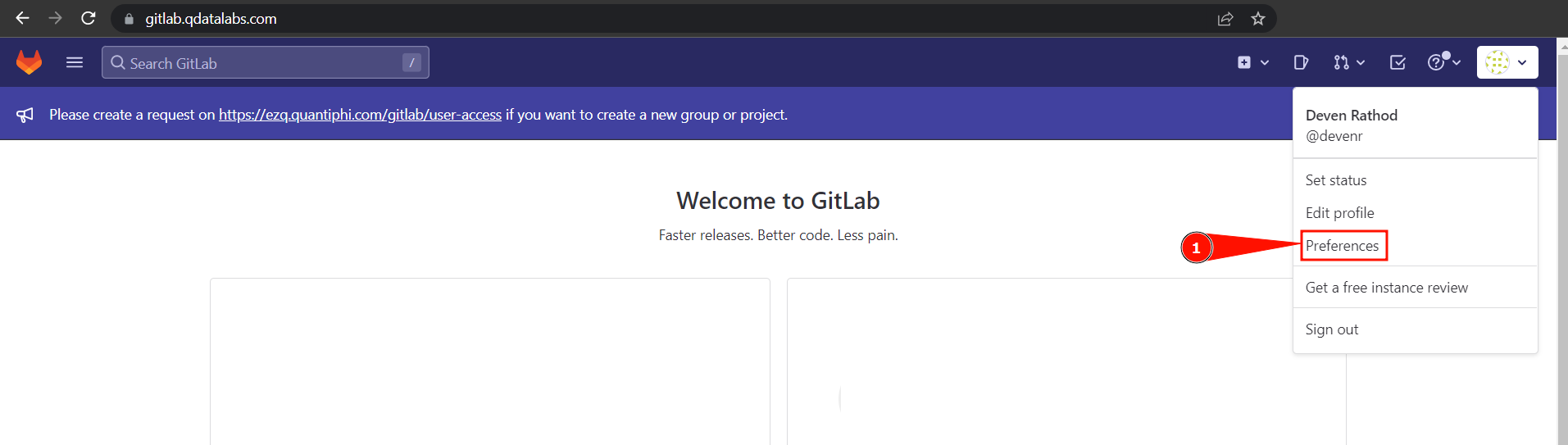
[**Creating Gitlab Access Token**](#_4ksqps6sfyfm)

[**Setting credentials in Jenkins before running the pipelines**](#_bd8r85s93cz7)

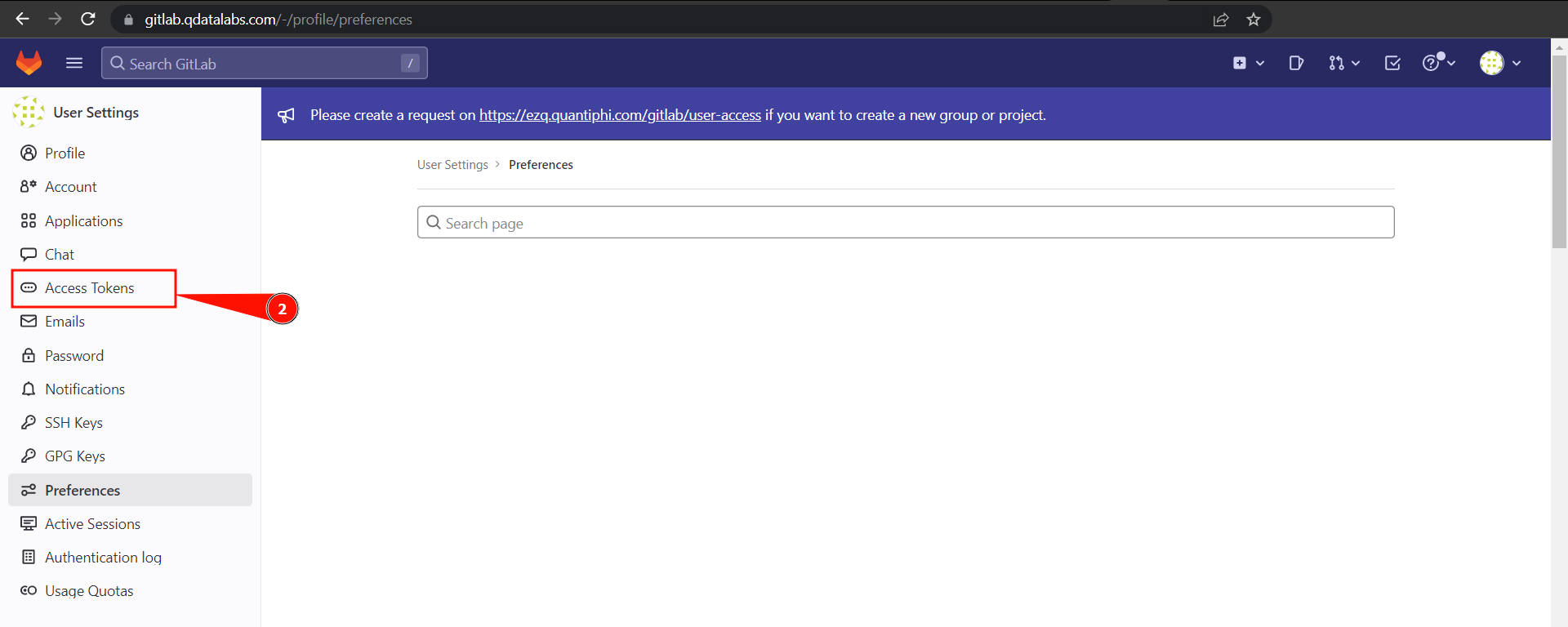
### **Creating Gitlab Access Token**

1. Login to Qdatalabs **Gitlab** (<https://gitlab.qdatalabs.com/>)

After logging in, on the right side you will see an option **Preferences**. Click on **Preferences** .



1. After selecting Preferences, on the left side you will see an option for **Access Tokens**. Click on **Access Tokens**.



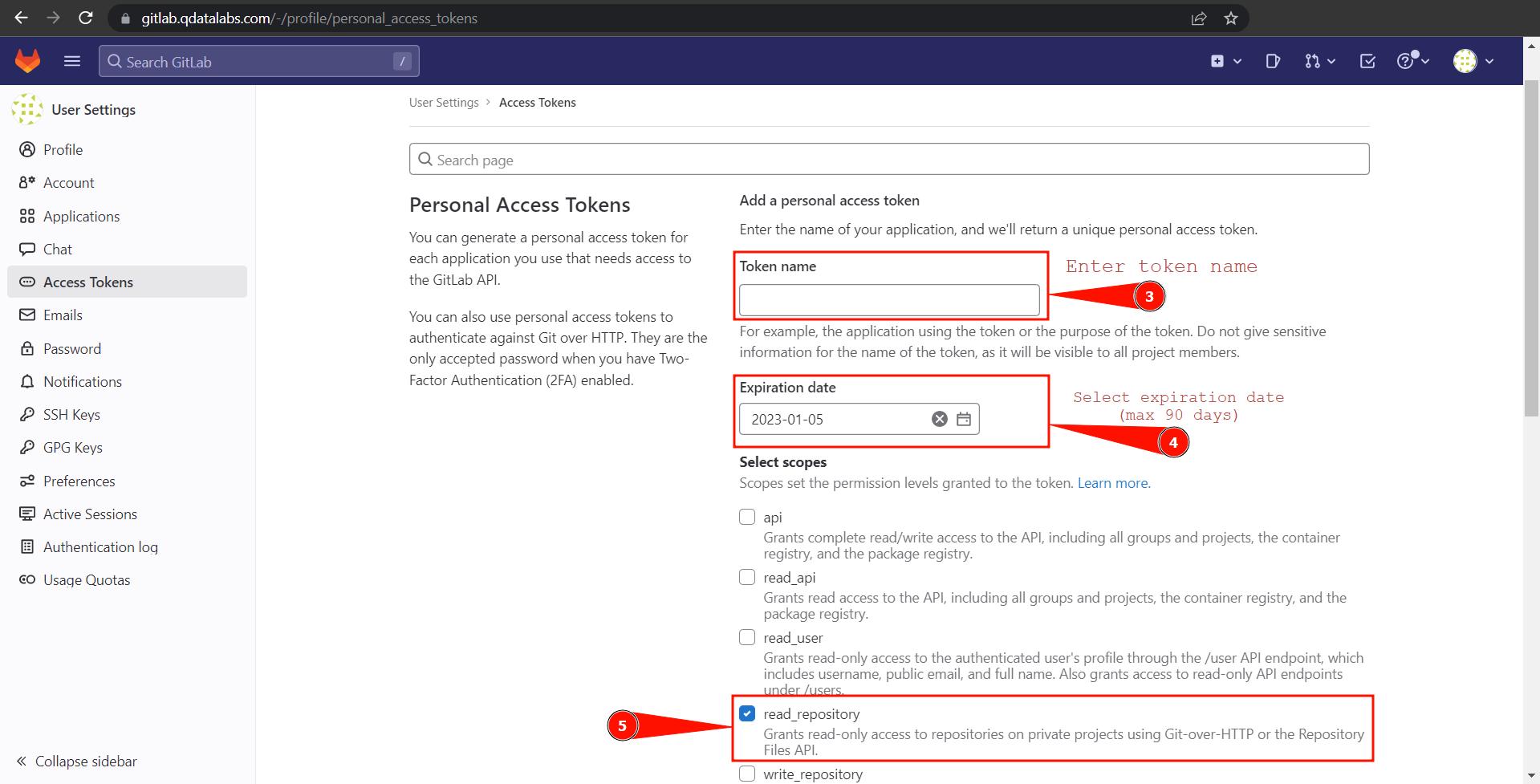
1. After selecting Access Tokens, Fill out the required fields

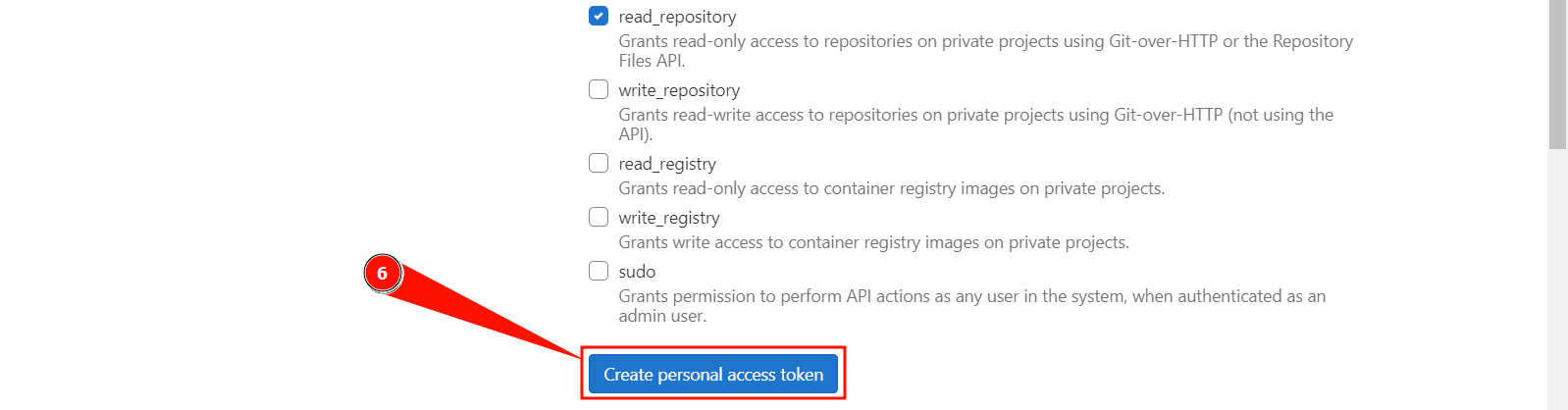
**Token name**: [*Enter token name*] *eg. projectname-devsecops*

**Expiration date:** [*Select expiration date for the access token*] (***90 days***)

**Select scopes:** [*Grant read-only access to the token to access*](**read\_repository**)

Click **Create personal access token** at the end

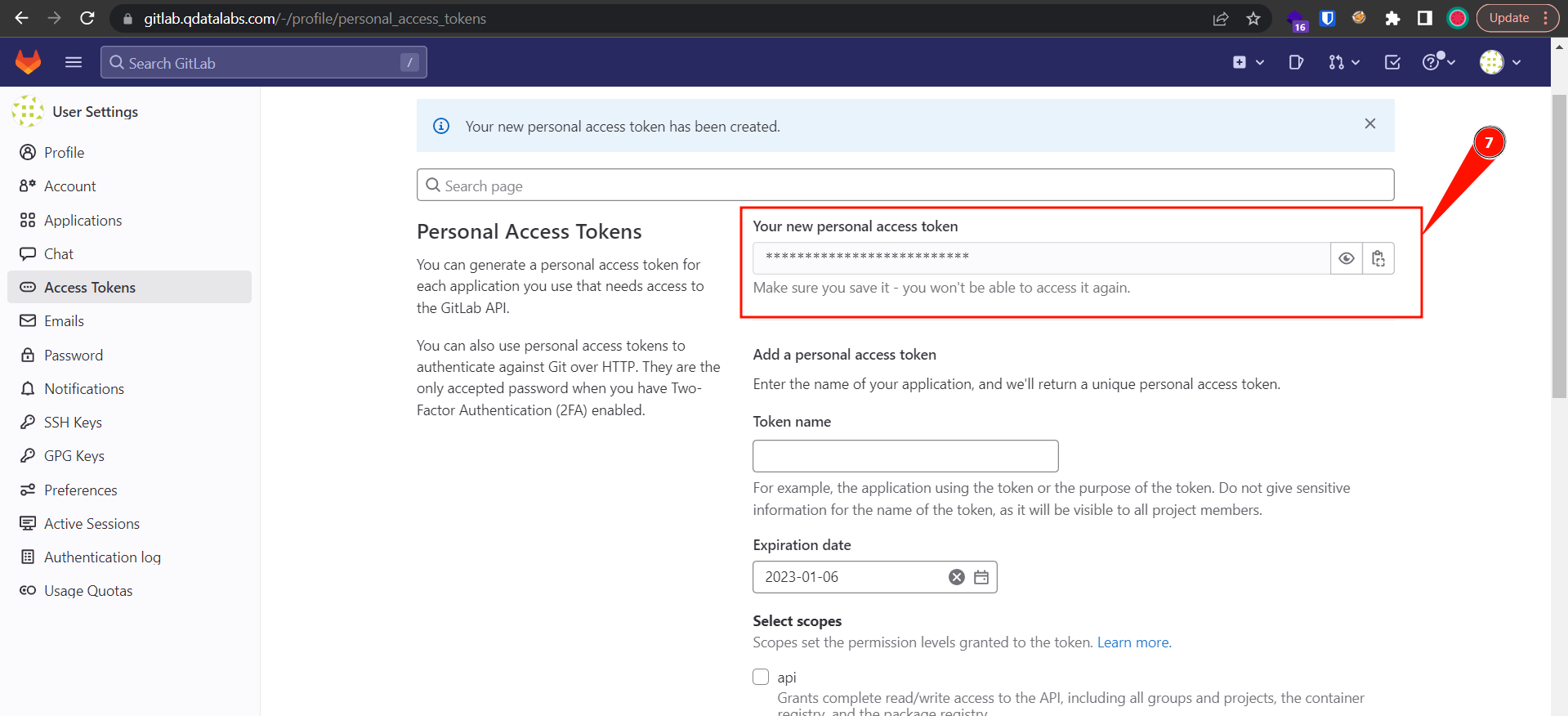




1. Scroll down and click on **Create personal access token**

**Make sure you safely save this generated access token displayed at the top**.

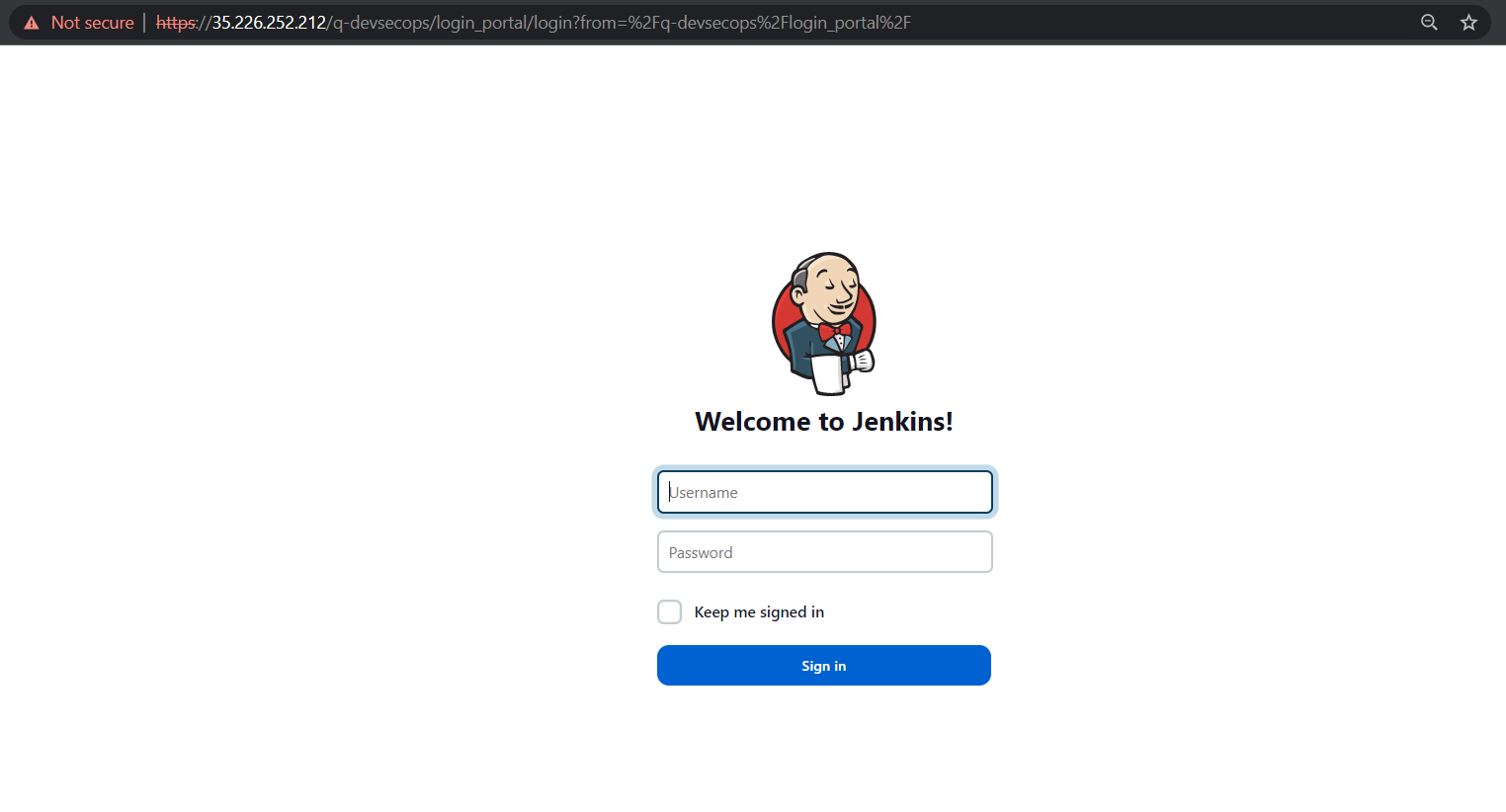
As shown below:



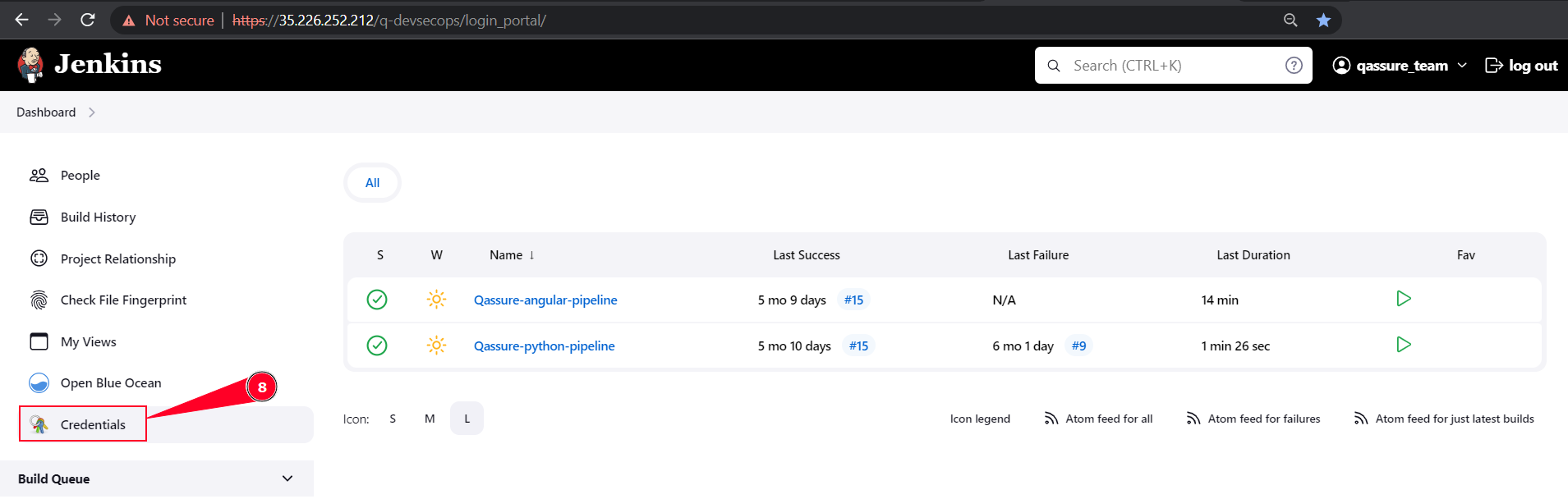
**Note:** **This same access token will be used as credentials on Jenkins in the password field to run the pipeline.**

### Setting credentials in Jenkins before running the pipelines

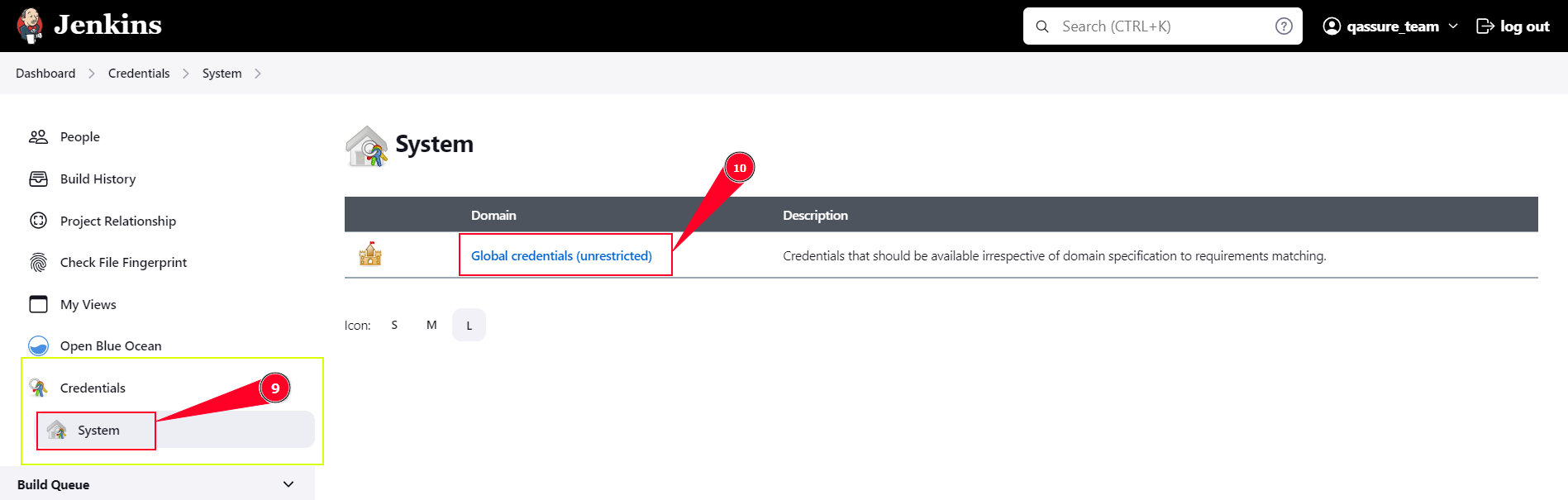
1. Login into DevSecOps **Jenkins** panel ([URL](https://35.226.252.212/q-devsecops/login_portal/))



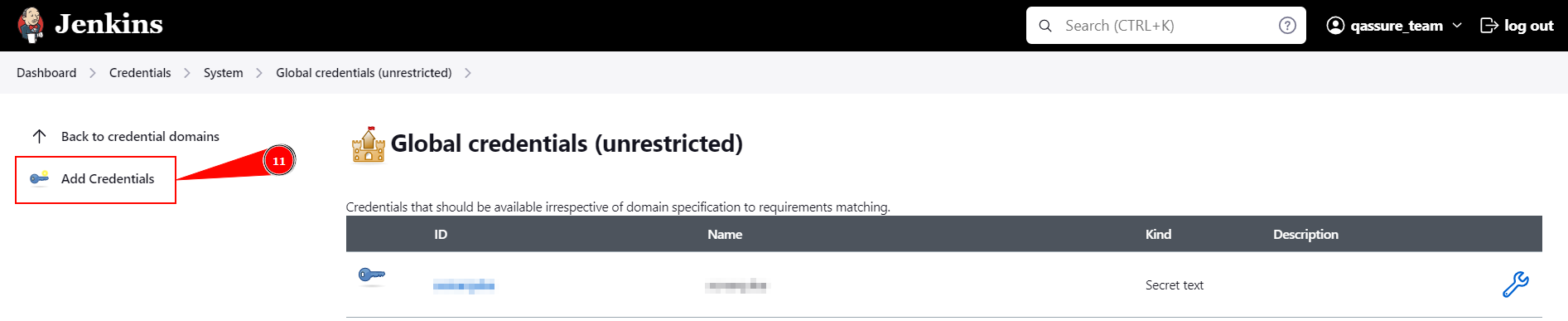
1. After login, on the left side you will see an option for **credentials**. Click on **Credentials**.



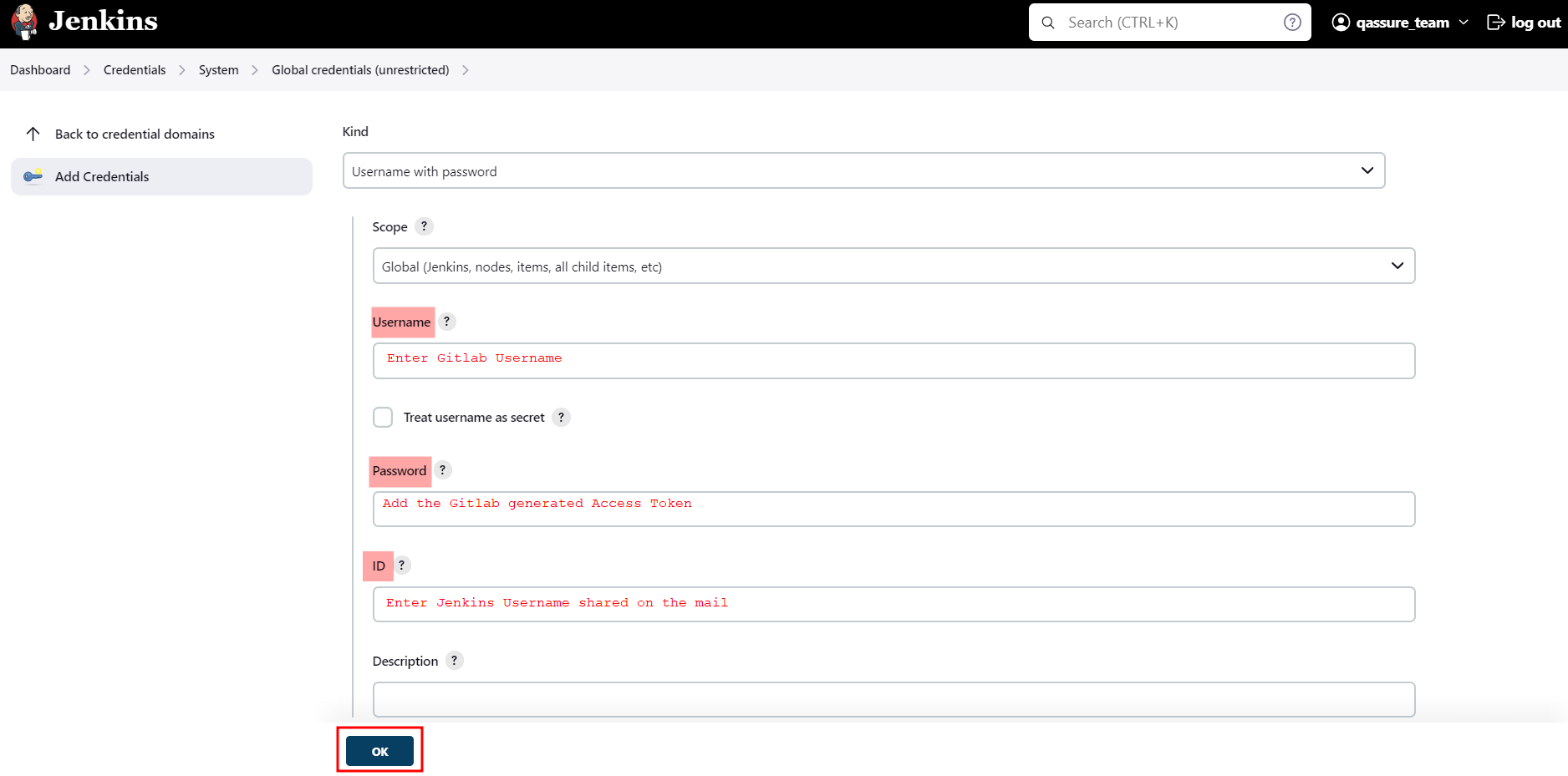
1. After selecting the credentials option, you will see a **System** option on the left side. Now, click on **System**



1. After selecting the System option, you will see an option for **Global Credentials**. Now, select the **Global Credentials**.
2. After selecting Global Credentials, on the left side you will see an option to **Add Credentials**. Click on the **Add Credentials**.



1. Fill out the details in the given fields.



In the **username and password field**, fill your qdatalabs **gitlab username** and in the password field add the generated access token to access the private repository.

**Username:** [*Gitlab username*]

**Password:** [*Access Token*]

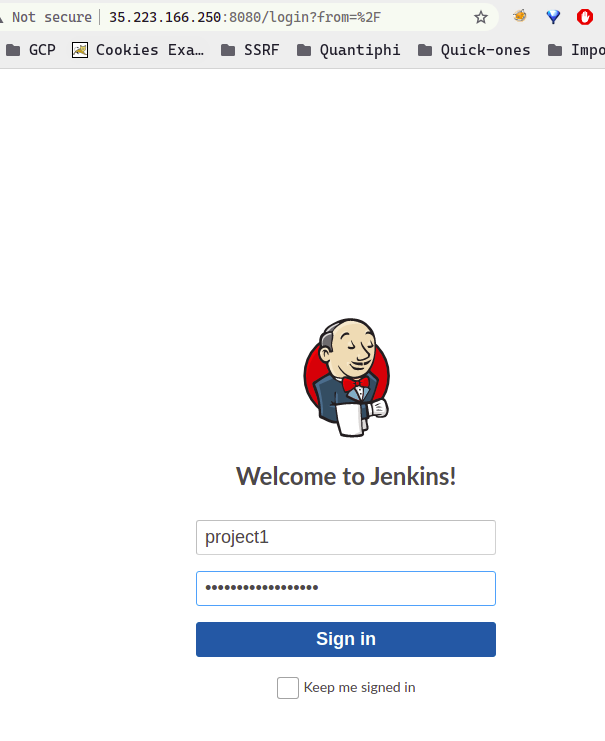
**ID:** [*Jenkins Username*]

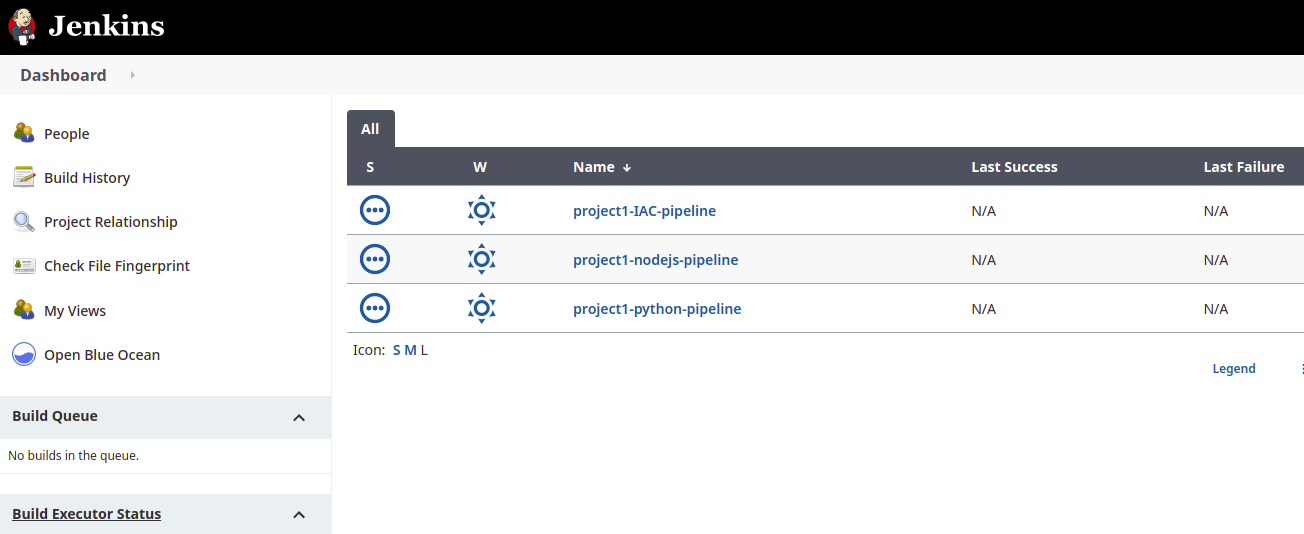
In the “ID” field, enter the Jenkins username used to login to Jenkins, provided in the mail and then click on **OK**.

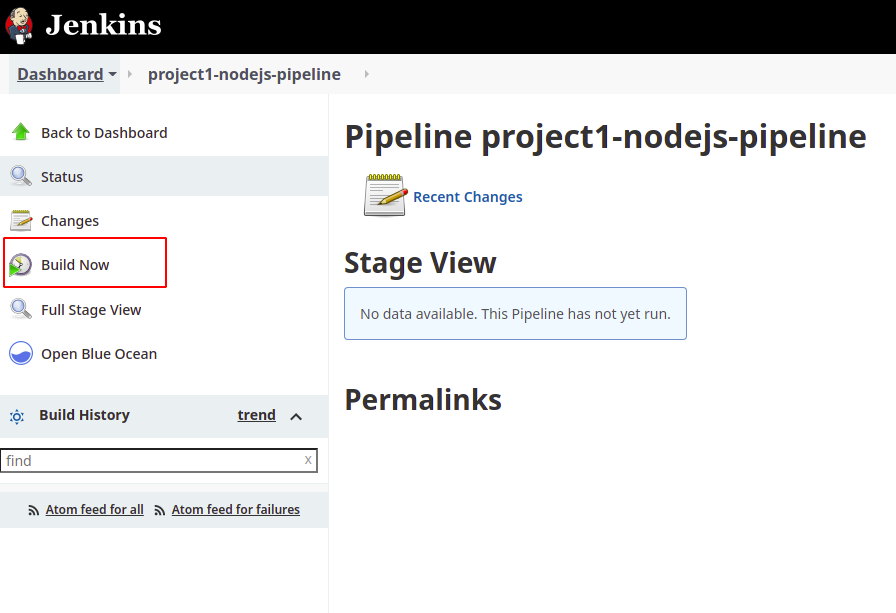
## 

## **How to use DevSecOps Pipeline?**

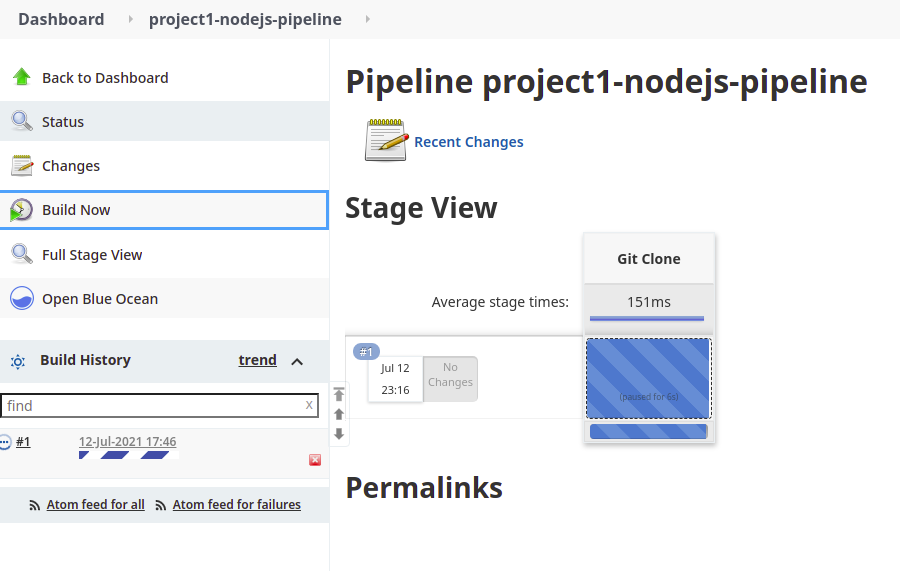
1. Visit the Jenkins URL [https://[IP]/q-devsecops/login\_portal/](https://35.226.252.212/q-devsecops/login_portal/)
2. Enter your credentials for login.
3. [Setup Credentials](#_keyipddlqdxh) (Gitlab credentials) in Jenkins



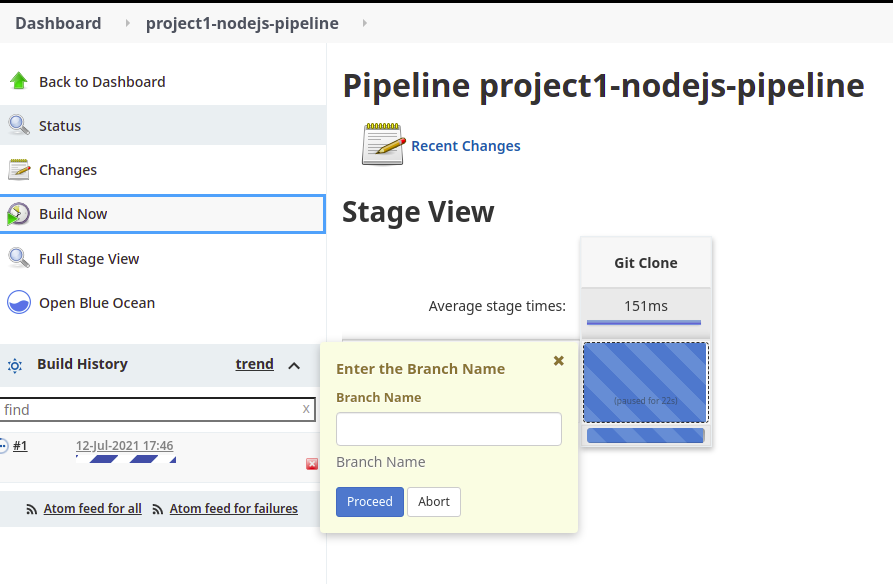
1. After login, you will see three pipeline with your project name e.g project1-IAC-pipeline 
2. These three pipelines are available for different languages e.g IAC, Nodejs and Python. If you are using Nodejs in your code then you can use Nodejs and If you are using IAC(terraform) then you can use IAC pipeline.
3. Click on your pipeline name based on language used in the project. On the left side, you will see an option to “**Build Now**”.



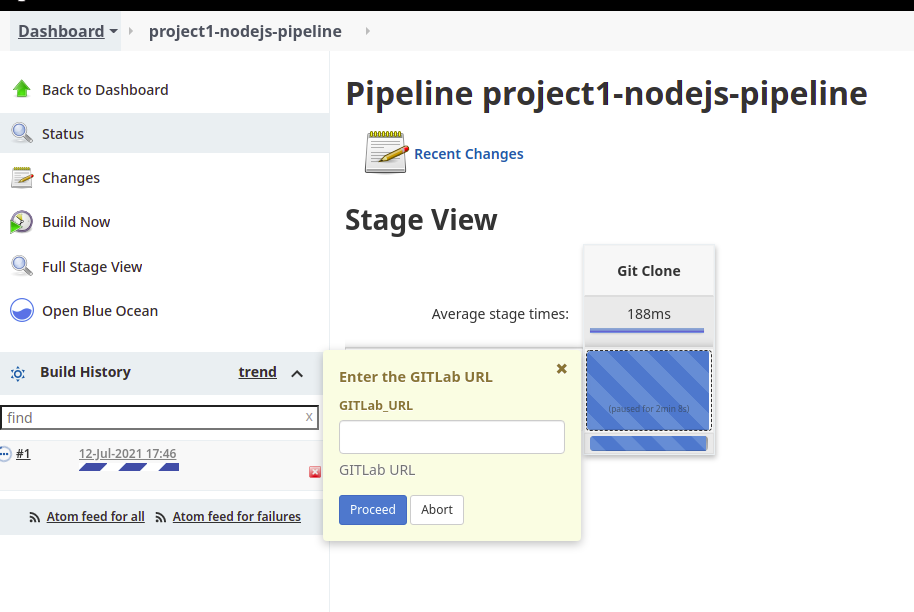
1. Click on “**Build Now**” and you will see that your pipeline has started. Now your pipeline will start as you can see in the screenshot below and it will wait for input from the user side



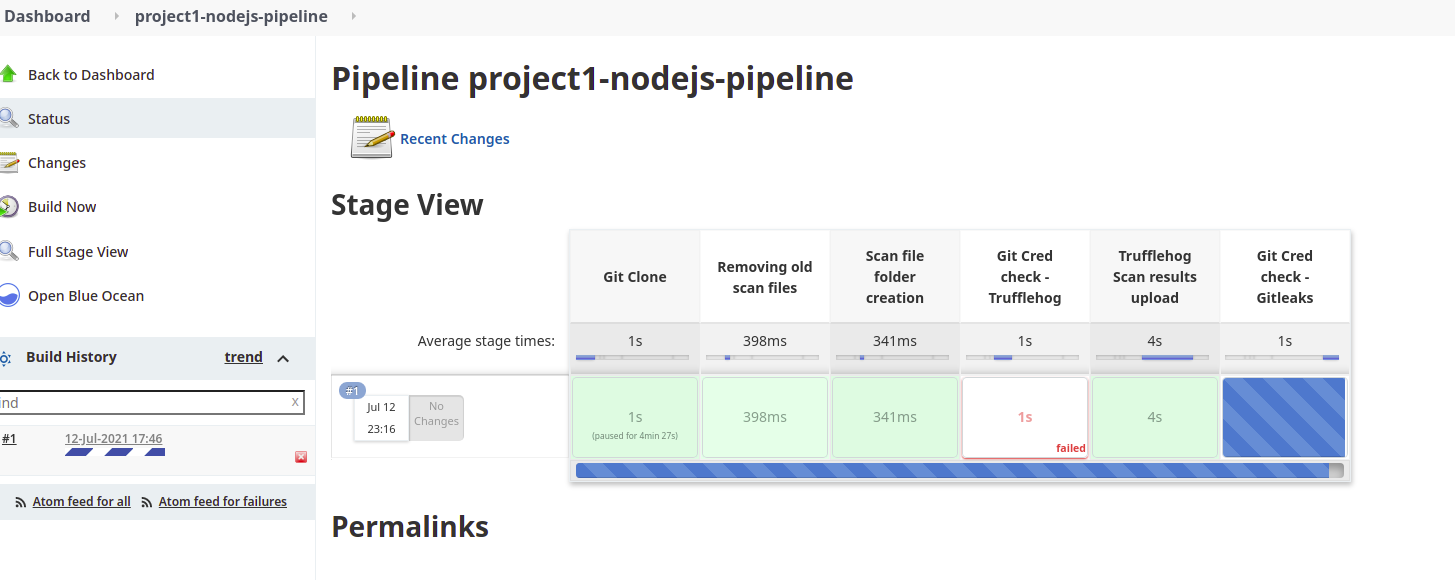
1. Click on the git clone stage and you will see an input. In input, we have to provide branch name and gitlab URL.



1. After providing Branch Name, you have to provide Gitlab URL. Provide the gitlab URL as per below screenshots and Click on **Proceed** after providing gitlab URL.

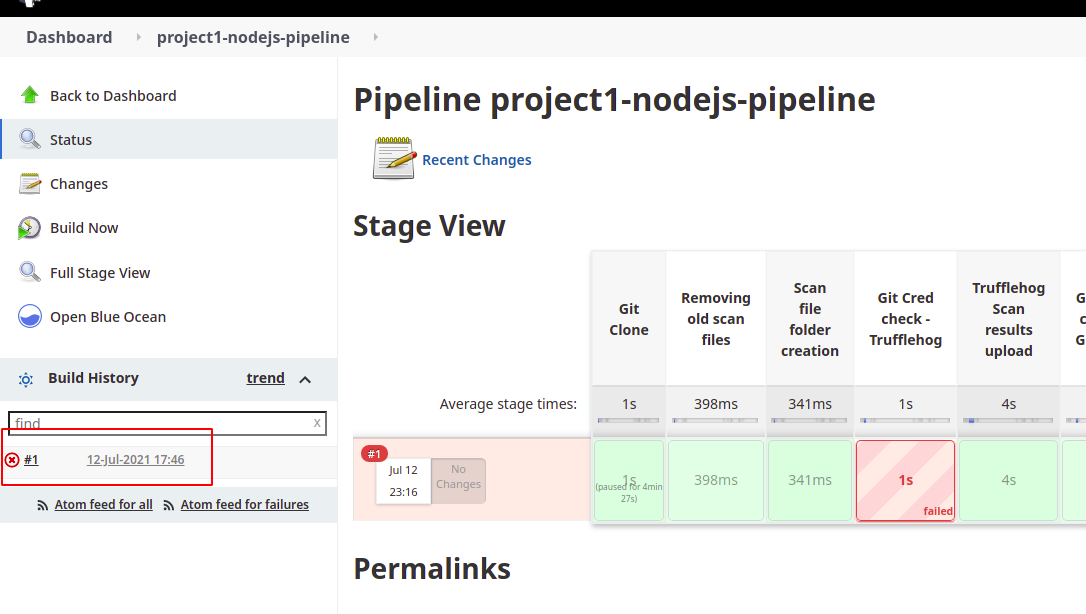


1. Pipeline will start performing the secure code review using the tools automatically.



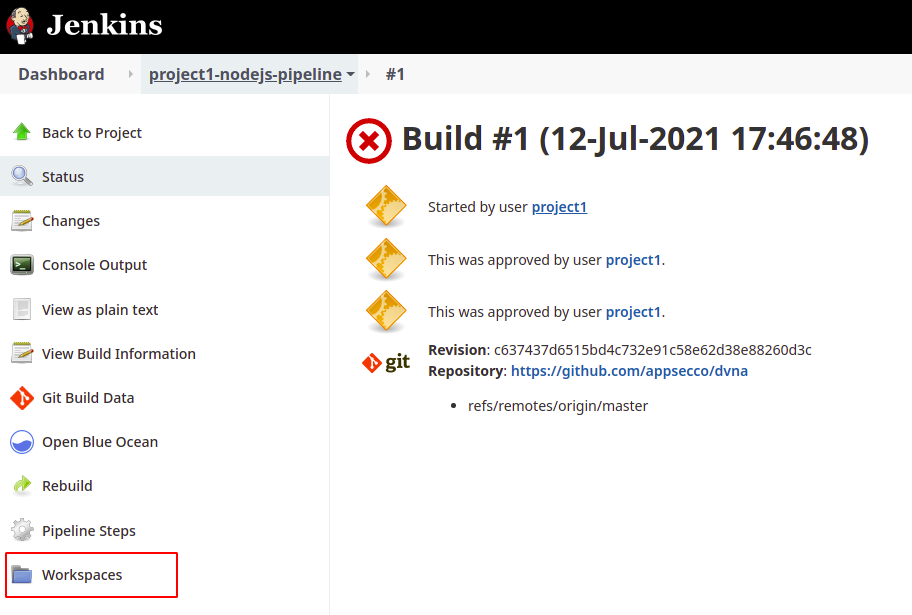
1. You can see the status of the pipeline in the pipeline flow. Pipeline will complete in 5-10 mins.
2. Pipeline will save the reports in a folder that you can view or download in zip format.

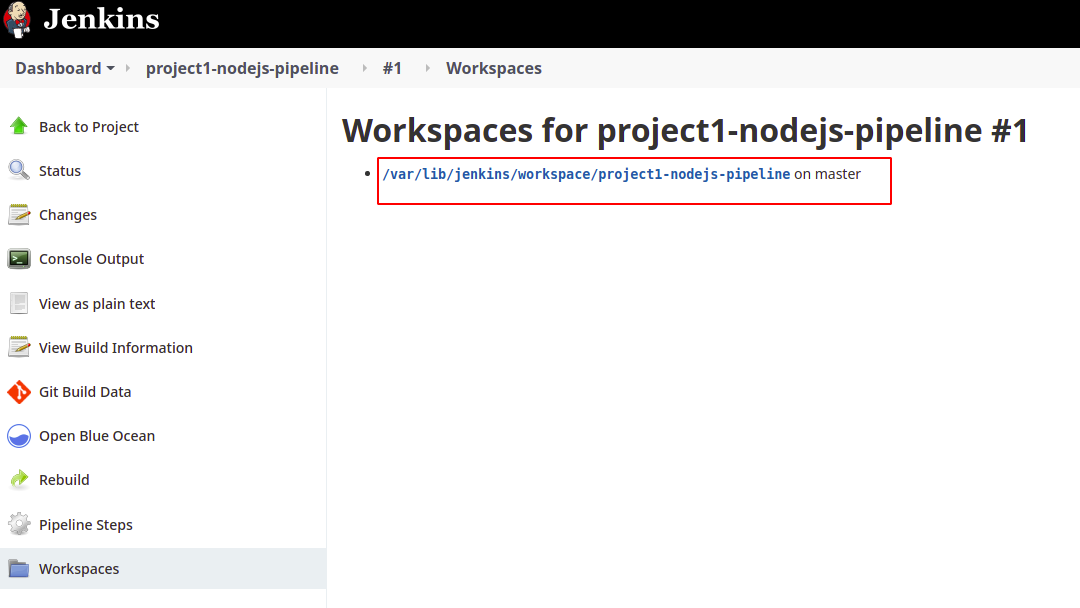
## **How to view/download pipeline results?**

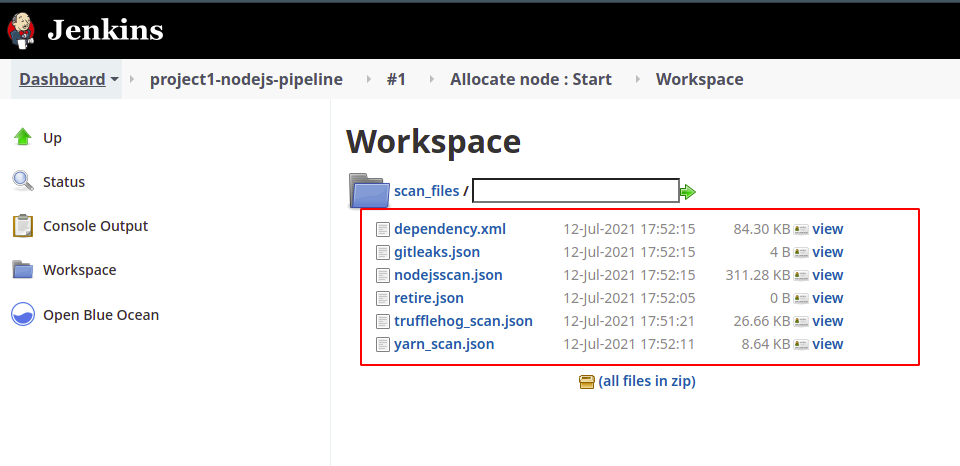
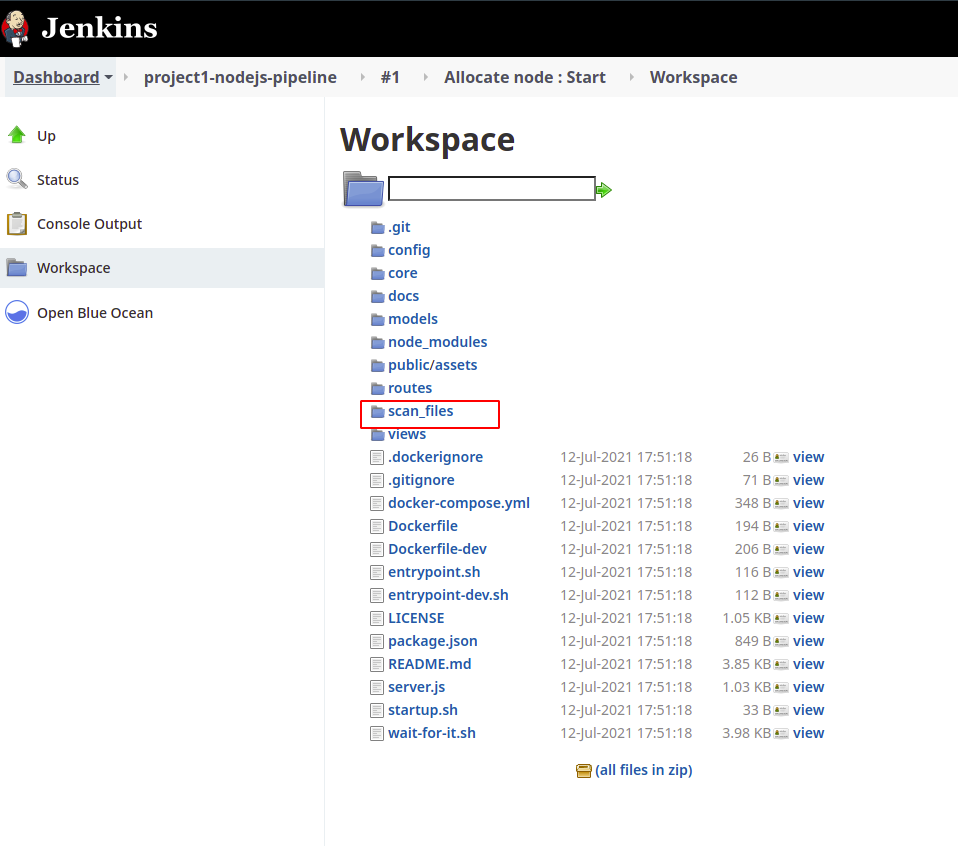
1. When the pipeline will finish, you can see reports from each tool separately.
2. To view the result of each tool, refer to this video.

<https://drive.google.com/file/d/1F04WZSX5WIJjn7bxMfusflWRzlVrMwoG/view?usp=sharing>

1. To Download the scan files, first click on your pipeline current build number
2. Then click on workspaces → click on folder → Click on “scan\_files folder” and you will be able to see







1. There is an option for “**all files in zip**”. You can select that option to download the zip file. After clicking on that, zip file will download in your system.

