DSAMS Web Application Architecture

**Version 1.0**

06/30/2021

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description of Version/Changes** | **Responsible Party** | **Date** |
| 1.0 | Initial version | Jack Zhen | 06/30/2021 |
| 1.1 | Review | Danial Beaver | 07/16/2021 |
| 1.2 | Review | Kerry Geist | 07/19/2021 |
| 1.3 | Review | Ruth Fritts |  |
| 1.4 | Review | Andy Brown |  |
|  |  |  |  |

**Table of Contents**

1. **Introduction**
   1. Purpose
   2. Scope
2. **Architectural Representation**
   1. DSAMS Web Application Workflow

2.1.1 Workflow Diagram

2.1.2 Infrastructure Diagram

* 1. DSAMS Web Application Security

2.2.1 Security Diagram

2.2.2 Authentication and Authorization

2.2.3 JSON Web Token (JWT)

2.2.4 Reference Links/Documents

* 1. DSAMS-Web Design

2.3.1 DSAMS-Web Source Code Infrastructure

2.3.2 The Angular 8 Framework

2.3.3 The Angular Basic Building Blocks

2.3.4 Reference Links/Documents

* 1. DSAMS-App Design

2.4.1 DSAMS-App Source Code Infrastructure

2.4.2 The Spring Boot 2.0 REST Service API

2.4.3 The JDBC/Data Source Connection

2.4.4 The RESTful Code

2.4.3.1 Representations

2.4.3.2 URLs/URL Structure

2.4.3.3 Standard Methods

2.4.3.4 REST API Actions

2.4.4 The Client/UI

2.4.4.1 Client and REST API Workflow Diagram

2.4.5 The Resources

2.4.5.1 JSON, XML, YMML Format

2.4.6 Reference Links/Documents

* 1. DSAMS-Legacy Implementation

2.5.1 Source Code

2.5.2 Build/Deploy/Implementation

1. **Source Code Developing**
   1. IDEs (Eclipse, VSCode, Toad)
   2. Languages (Java 11, Angular 8, HTML 5)
2. **Source Code Control**
   1. DSAMS Web Application in GitLab Diagram
   2. DSAMS Branch/Feature
3. **Source Code Quality** 
   1. Code Review
   2. Junit Test
   3. Integration Test
   4. SonarQube
   5. Twistlock
4. **Data View**
   1. DSAMS Database Diagram
5. **Deployment View**
   1. Jenkins Build Process States Diagram
   2. Jenkins
   3. Docker
6. **Reference**
7. **Introduction**

This document provides a high level overview and explains the architecture of the Distributed Development DSAMS Web Application.

The document defines goals of the architecture and components that have been selected. The document provides a rationale for the architecture and design decisions made from the conceptual idea to its implementation

1.1 Purpose

The Architecture Document provides a comprehensive architectural overview of the DSAMS Web Application. It presents a number of different architectural views to depict the different aspects of the system

1.2 Scope

The scope of this Architecture Document is to explain the architecture of the DSAMS Web Application.

This document describes the various aspects of the DSAMS Web Application design that are considered to be architecturally significant. These elements and behaviors are fundamental for guiding the construction of the DSAMS Web Application and for understanding this project as a whole.

1. **Architectural Representation**
   1. DSAMS Web Application Workflow

2.1.1 Workflow Diagram

[Spring Boot](https://www.baeldung.com/spring-boot) and [Angular](https://angular.io/) form a powerful tandem that works great for

developing web applications with a minimal footprint.

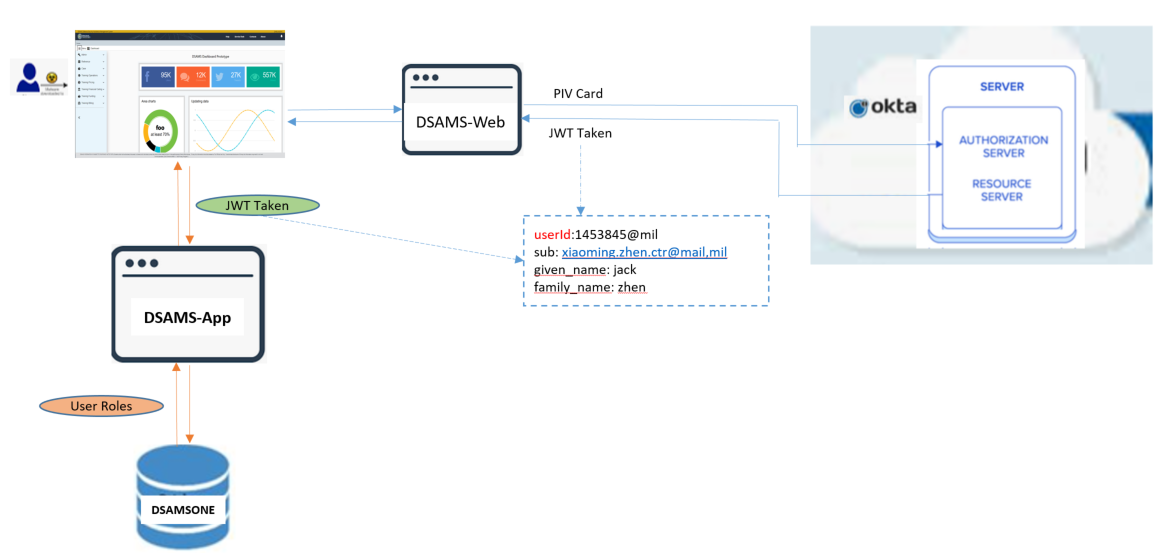
DSAMS Web Application is a three tiers application. It includes User Interface

(Angular), Middle Tier (RESTFul API) and Oracle Database. When the user entry

the information from the UI and execute the commands (CRUD). The RESTFul

API will response the user request (CRUD) from the Database, then getting the

results to the user by the URI.



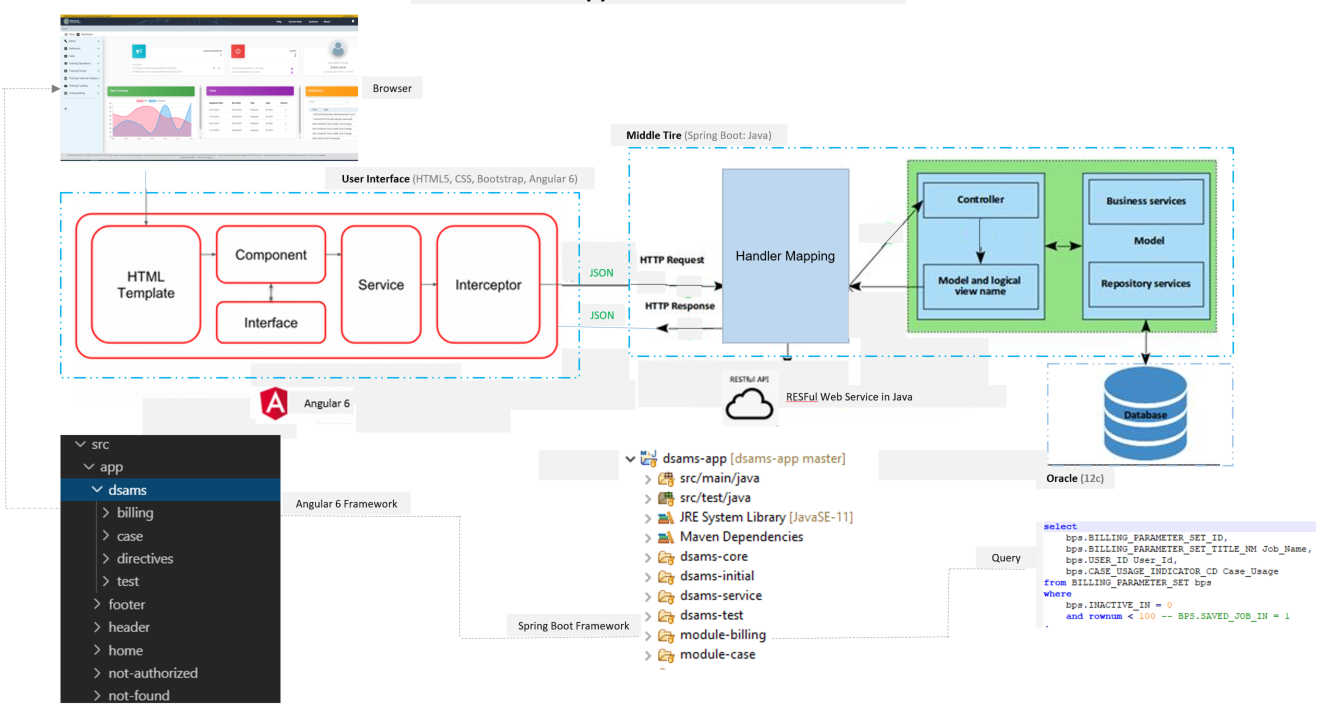
2.1.2 Infrastructure Diagram

DSAMS Web is a three tiers application. It includes

1. User Interface which is created by Angular 8, JavaScript, HTML 5

and CSS. It provides interactions between users and DSAMS Web occur.

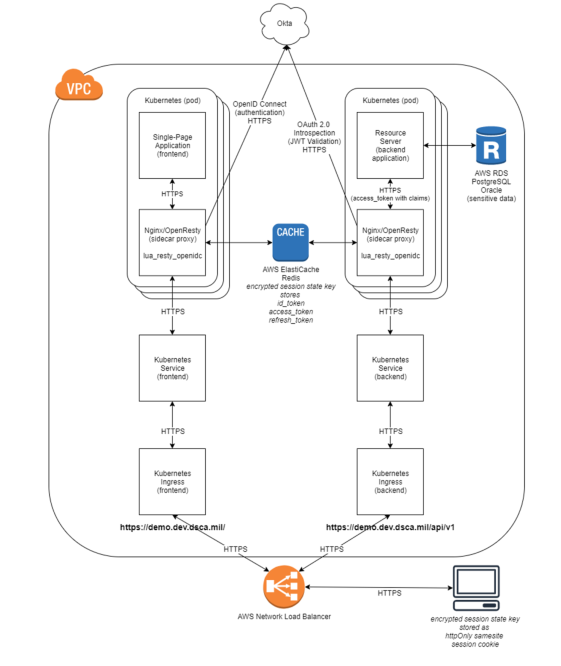
1. Middle Tier which is created by Java Spring Boot framework is an architectural style for an application program interface (API) that uses HTTP requests to access and use data.
2. Database is Oracle 19c which store all the DSAMS data information.



* 1. DSAMS Web Application Security

2.2.1 DSAMS Security Diagram

Authentication for frontend and authorization for backend applications is provided through the use of **Okta** SaaS (Software as a Service). The DSCA has procured and deployed [Okta](https://www.okta.com/) as a solution for single sign-on. Okta is a solution that is [FedRAMP](https://www.fedramp.gov/) certificated and supports [OpenID Connect](https://openid.net/connect/) authentication via CAC/PIV in addition to SAML and [OAuth 2.0](https://oauth.net/2/) authorization. Okta integrates with all of the third-party tools the DSCA has procured such as Jira, Jenkins, Confluence, etc for the modernization effort. Detailed information on this topic may be found at the following link: [Enterprise Authentication and Authorization Strategy](https://apps.dsca.mil/wiki/display/FSC/Enterprise+Authentication+and+Authorization+Strategy)



2.2.2 JSON Web Token (JWT)

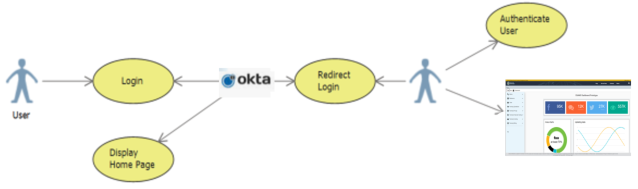
The access token in this case is also referred to as a bearer token. It is a **JSON Web Token** (JWT) which contains claims that the user has been assigned within Okta. These claims can be groups, countries or other attributes that your application requires to perform role based and fine grained (or attribute based) access control (RBAC and ABAC respectively)

JWT Token Example:





2.2.3 User Login Diagram

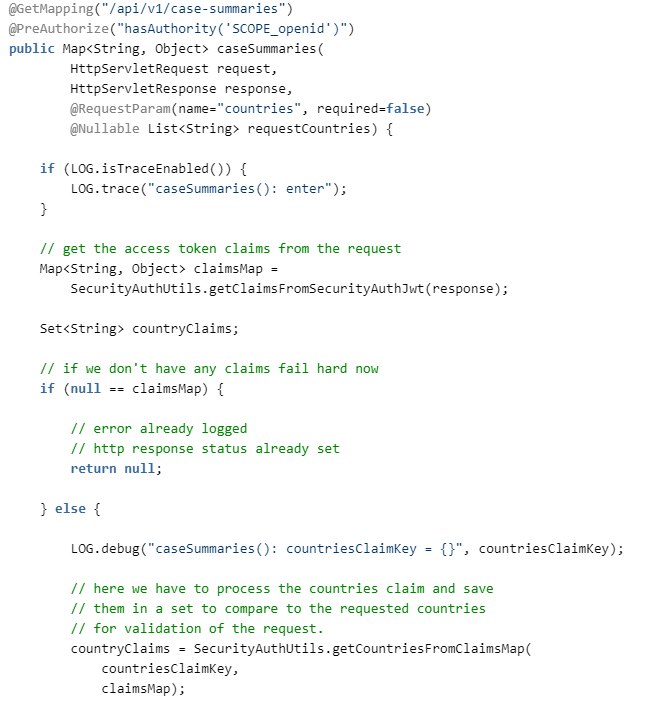


2.2.4 The following is a snippet of

a) Get Global withCredentials (Angular)



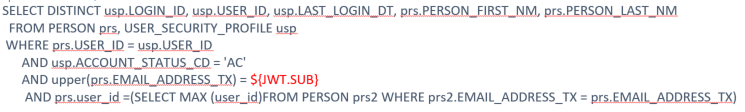
b) Get Information from JWT (Java)



2.2.4 Authentication and Authorization

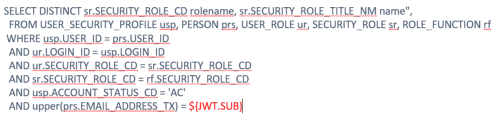
1. User Authentication

Retrieve the user credential from **JWT Token** and use it to query the user if the user exists in the database.



1. User Authorization

Retrieve the user credential from **JWT Token** and use it to query the user’s authoritation in the database.



2.2.5 Reference Links/Documents

a) [DSAMS Web Application Authentication and Authorization](https://apps.dsca.mil/wiki/pages/viewpage.action?spaceKey=FSC&title=Write+Code#WriteCode-AuthenticationandAuthorization)

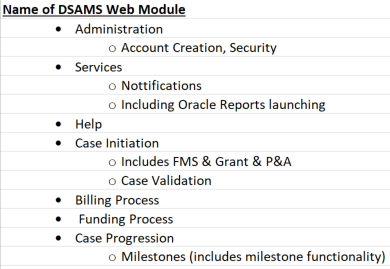
b) [Spring REST API OAuth2 Angular](https://www.baeldung.com/rest-api-spring-oauth2-angular)

c) [OKTA Developer](https://developer.okta.com/)

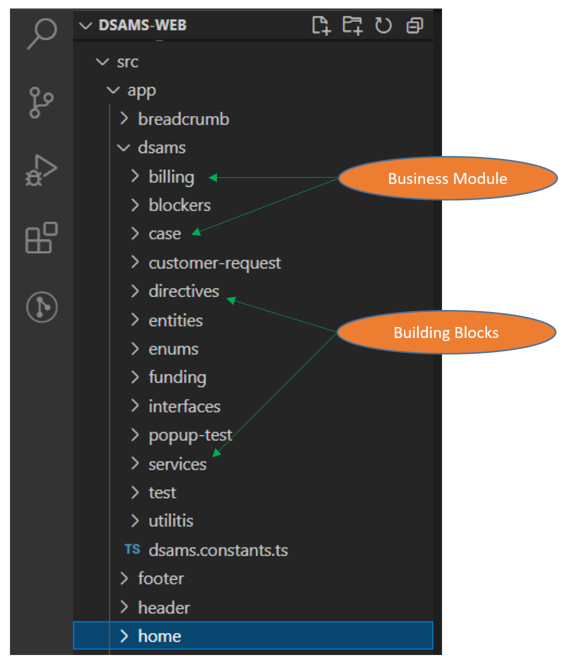
* 1. DSAMS-Web Design

2.3.1 DSAMS-Web Source Code Infrastructure

1. Dependency on DSAMS Business Module



b) DSAMS-Web Project Structure in VSCode 1.5



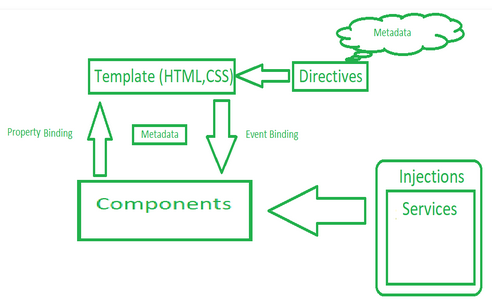
2.3.2 The Angular 8 Framework

**Angular is a framework** for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript.

The framework consists of several libraries, some of them core and some optional.

2.3.3 The Angular Basic Building Blocks

1. Basic Building Blocks diagram



1. Modules

Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.

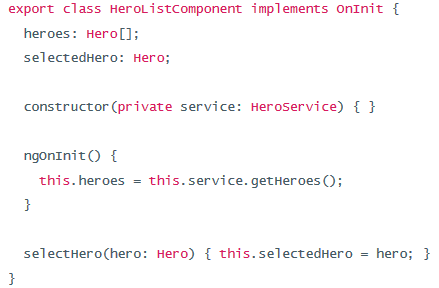
Simple Snippet Code:



1. Components

Every angular app has at least 1 component known as the root component that connects a component hierarchy with the page document object model (DOM). Each component defines a class containing the login in TypeScript format and view of the page in HTML template

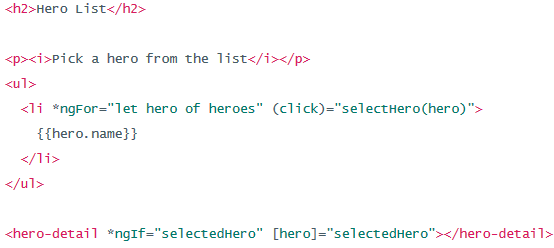
Simple Snippet Code:



1. Templates

The HTML templates have the look of how the page will look. The template logic provided by directives and the linking of the application data with the DOM of the page is done by using binding

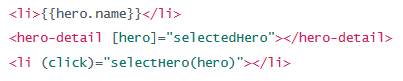
Simple Snippet Code:



1. Data Binding

Data binding is used to pass data from component class and facilitates interpolation of values that are computed from the application back into the HTML

Simple Snippet Code:





1. Directives

Directives transforms the DOM according to the instructions.

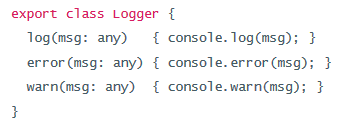
Simple Snippet Code:



1. Services

Service is a broad category encompassing any value, function, or feature that your application needs

Simple Snippet Code:



1. Dependency Injection

Dependency Injection allow access to the service by subscribing it. It acts as a delegate to the service

Simple Snippet Code:



2.3.4 Reference

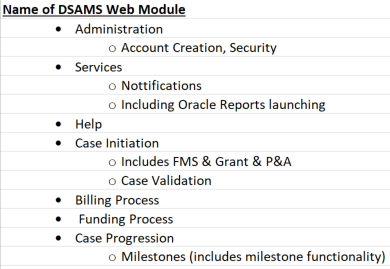
a) Angular 8: <https://angular.io/>

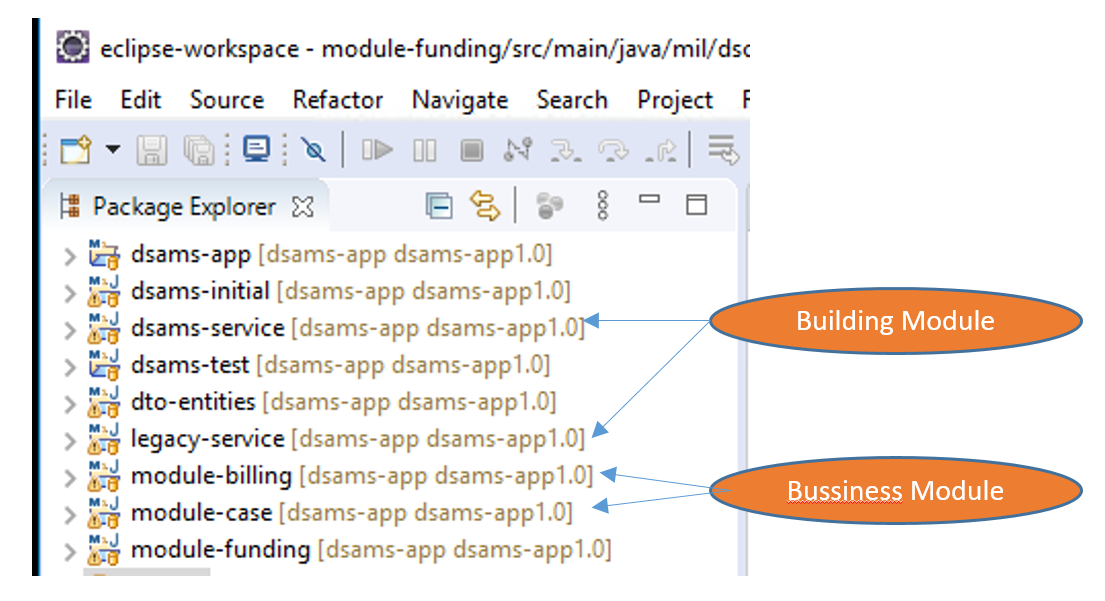
b) Material Design: <https://material.io/design/>

* 1. DSAMS-App Design

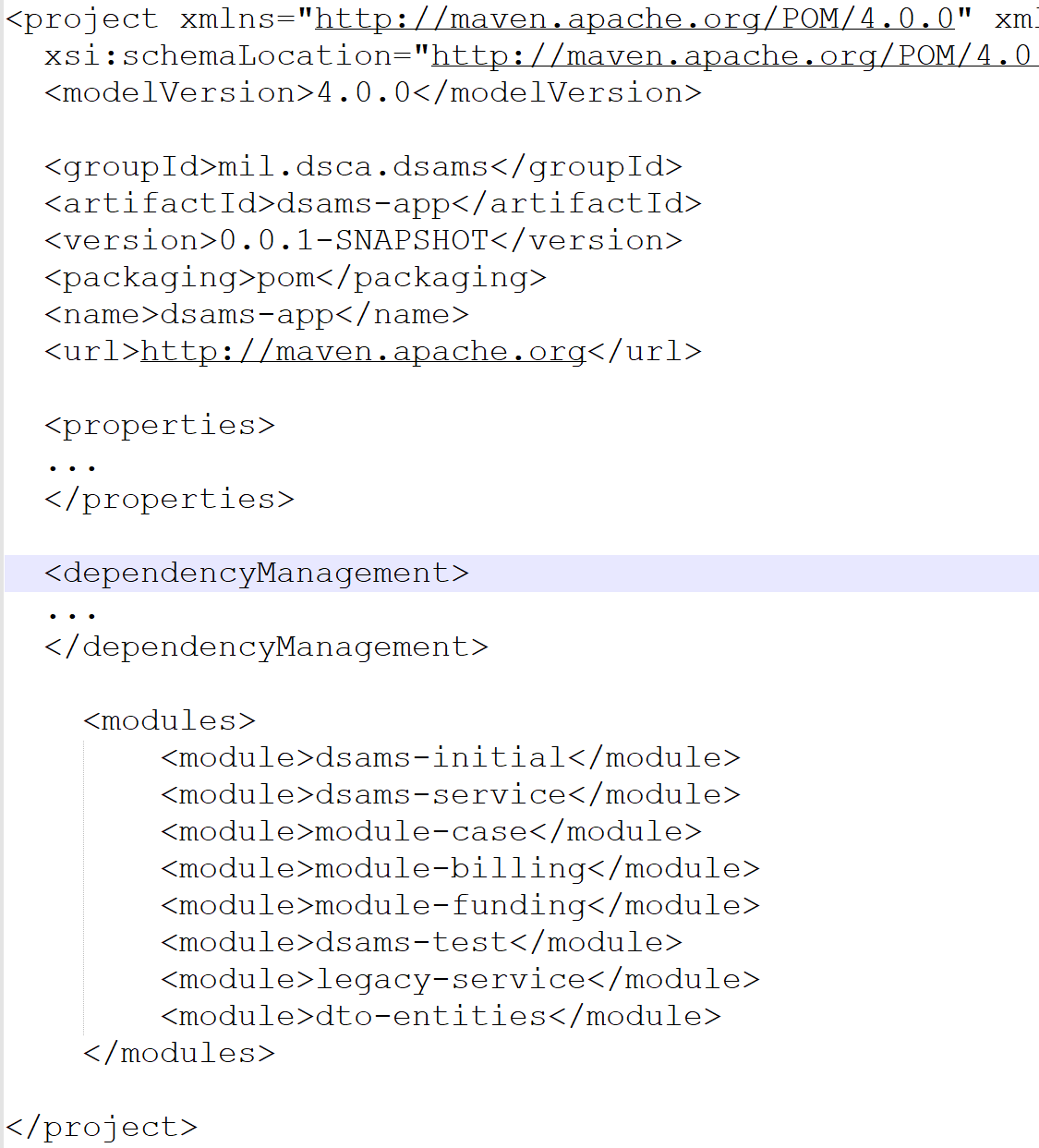
2.4.1 DSAMS-App Source Code Infrastructure

a) Dependency on DSAMS Business Module



1. DSAMS-App Project structure in Eclipse 4.1 

c) Project Module Dependency in POM.xml



2.4.2 The Spring Boot 2.0 REST Service API

A **REST API** is an intermediary Application Programming Interface that enables two applications to communicate with each other over HTTP, much like how servers communicate to browsers.

2.4.3 The JDBC/Data Source Connection

**Spring Boot JDBC** provides starter and libraries for connecting an application with JDBC.In Spring Boot JDBC, the database related beans such as **DataSource, JdbcTemplate**, and **NamedParameterJdbcTemplate** auto-configures and created

during the startup.

**Connection String**:

<Resource

auth="Container"

driverClassName="oracle.jdbc.OracleDriver"

factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"

initialSize="3"

maxActive="5"

maxIdle="3"

maxWait="5000"

minEvictableIdleTimeMillis="60000"

minIdle="1"

name="jdbc/rfdbDS"

numTestsPerEvictionRun="5"

password="\*\*\*\*\*"

testOnBorrow="true"

testWhileIdle="true"

timeBetweenEvictionRunsMillis="30000"

type="javax.sql.DataSource"

url="jdbc:oracle:thin:@dev-dsams-rds-cd3.c1pbhwdyaf37.us-gov-west-1.rds.amazonaws.com:1521:DSAMSCD3"

username="SANDBOX\_APP[DSAMSRFDB\_APP]"

validationQuery="SELECT 'OK' FROM dual"/>

2.4.4 The REST Service API Snippet Code

2.4.3.1 Representations

The **Entity Layer** which represents the data to the user.

public class PostDto {

private static final SimpleDateFormat dateFormat

= new SimpleDateFormat("yyyy-MM-dd HH:mm");

private Long id;

private String title

}

The **Service** Layer which operates the business logic.

public List<Post> getPostsList(

int page, int size, String sortDir, String sort) {

PageRequest pageReq

= PageRequest.of(page, size, Sort.Direction.fromString(sortDir), sort);

Page<Post> posts = postRepository

.findByUser(userService.getCurrentUser(), pageReq);

return posts.getContent();

}

The **Controller Layer** which implements REST API for the POST resource.

@Controller

class PostRestController {

@Autowired

private IPostService postService;

@Autowired

private IUserService userService;

@Autowired

private ModelMapper modelMapper;

}

The **Repository Layer** which retrieves the information from the database.

@Repository

class PostRestRepository {

public findByUserId(int userId) {

String sql = String.join("\n",

"Select LOGIN\_ID from USER\_SECURITY\_PROFILE where userId = ?");

Object[] sqlParameters = new Object[] { userId };

try {

jdbcTemplateSec.update(sql, sqlParameters);

} catch (Exception ex) {

log.error("Exception thrown findByUserId(): ", ex);

}

}

}

2.4.3.2 URLs/URL Structure

a) Entry Point

A RESTful API needs to have one and exactly one entry point. The URL of the entry point needs to be communicated to API users so that they can find the API.

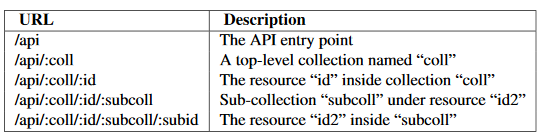
Technically speaking, the entry point can be seen as a singleton resource that exist outside any collection. It is common for the entry point to contain some or all of the following information:

• Information on API version, supported features, etc.

• A list of top-level collection.

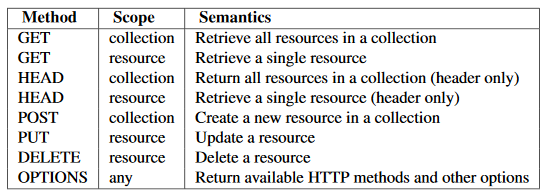
• A list of singleton resources.

b) URL Structure



2.4.3.3 Standard Methods

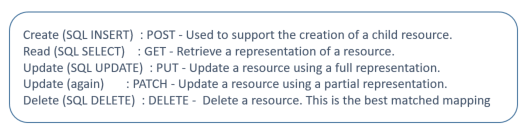
The table below lists the standard methods that have a well-defined meaning for all resources and collections.



2.4.3.4 REST API Actions

RESTful operations, an “actions” sub-collection can be used on a resource. Actions are basically RPC-like messages to a resource to perform a certain operation. The “actions” sub-collection can be seen as

a command queue to which new action can be POSTed, that are then executed by the API. Each action resource that is POSTed, should have a “type” attribute that indicates the type of action to be performed, and can have arbitrary other attributes that parametrize the operation.

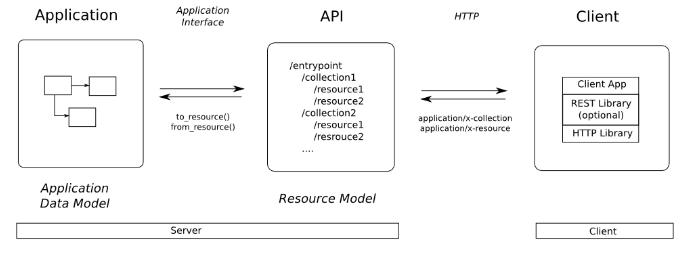


2.4.4 The Client & Rest API

The client consumes the RESTful API via the standard HTTP protocol. In theory, the service could be provided on top of other protocols as well.

Clients would typically use an HTTP library to access the RESTful API.

Client and REST Service API Workflow Diagram



2.4.5 The Resources

1. JSON Format

{

"\_type": "vm",

"name": "A virtual machine",

"memory": 1024,

"cpu": {

"cores": 4,

"speed": 3600

},

"boot: {

"devices": ["cdrom", "harddisk"]

}

}

1. XML

<xml xmlns:xs="http://www.w3.org/2001/XMLSchema">

<name type="xs:string">My VM</name>

<memory type="xs:int">1024</memory>

<cpu>

<cores type="xs:int">4</cores>

<speed type="xs:int">3600</speed>

</cpu>

<boot>

<devices type="xs:list">

<device type="xs:string">cdrom</device>

<device type="xs:string">harddisk</device>

</devices>

</boot>

</xml>

1. YAML Format

!vm

name: A virtual machine

memory: 1024

cpu:

cores: 4

speed: 3600

boot:

devices:

- cdrom

- harddisk

2.4.6 Reference Links/Documents

a) Spring Boot – Build REST API: [https://Spring Boot](https://www.tutorialspoint.com/spring_boot/spring_boot_building_restful_web_services.htm)

b) REST Service API: <https://spring.io/guides/gs/rest-service/>

c) JSON: <https://www.w3schools.com/js/js_json_intro.asp>

* 1. DSAMS Legacy implementation

To speed the development of the DSAMS App. We implement part of the legacy code from the DSAMS-Core. In order to use the legacy code in the

DSAMS App. We make the legacy code as module from the DSAMS App

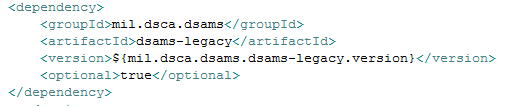
project. The legacy code will be built by Maven into a jar file and stores into

artfactory repository. The legacy jar file will be pull into the DSAMS App.

during DSAMS App. build for the Jenkins.

1. Legacy module dependency in DSAMS App POM.XML

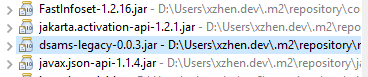




1. Artfactory repository ULR

<https://apps.dsca.mil/artifactory/maven/mil/dsca/dsams/dsams-legacy/>

1. Legacy Jar file in Maven dependency

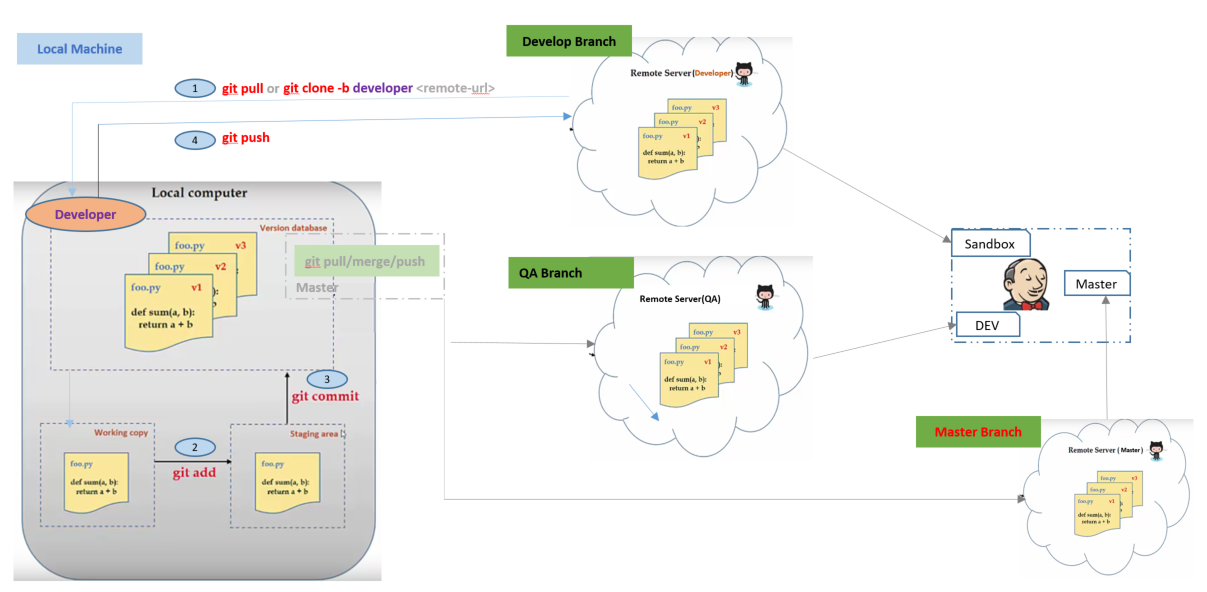


1. **Source Code Developing**
   1. IDEs
2. For User Interface: MS Visual Studio Code
3. For Middle Tier: Eclipse 4.14
4. Database: Toad/SQL Developer
   1. Languages & Scripts
5. Java, Typescript, JavaScript, HTML, CSS
6. **Source Code Control**
   1. DSAMS Web Application in GitLab Diagram

The DSAMS web application source code is stored in the GitLab.

GitLab is a web-based DevOps lifecycle tool that provides a Git repository manager providing wiki, issue-tracking and continuous integration and deployment.

Source code workflow in GitLab disgarm

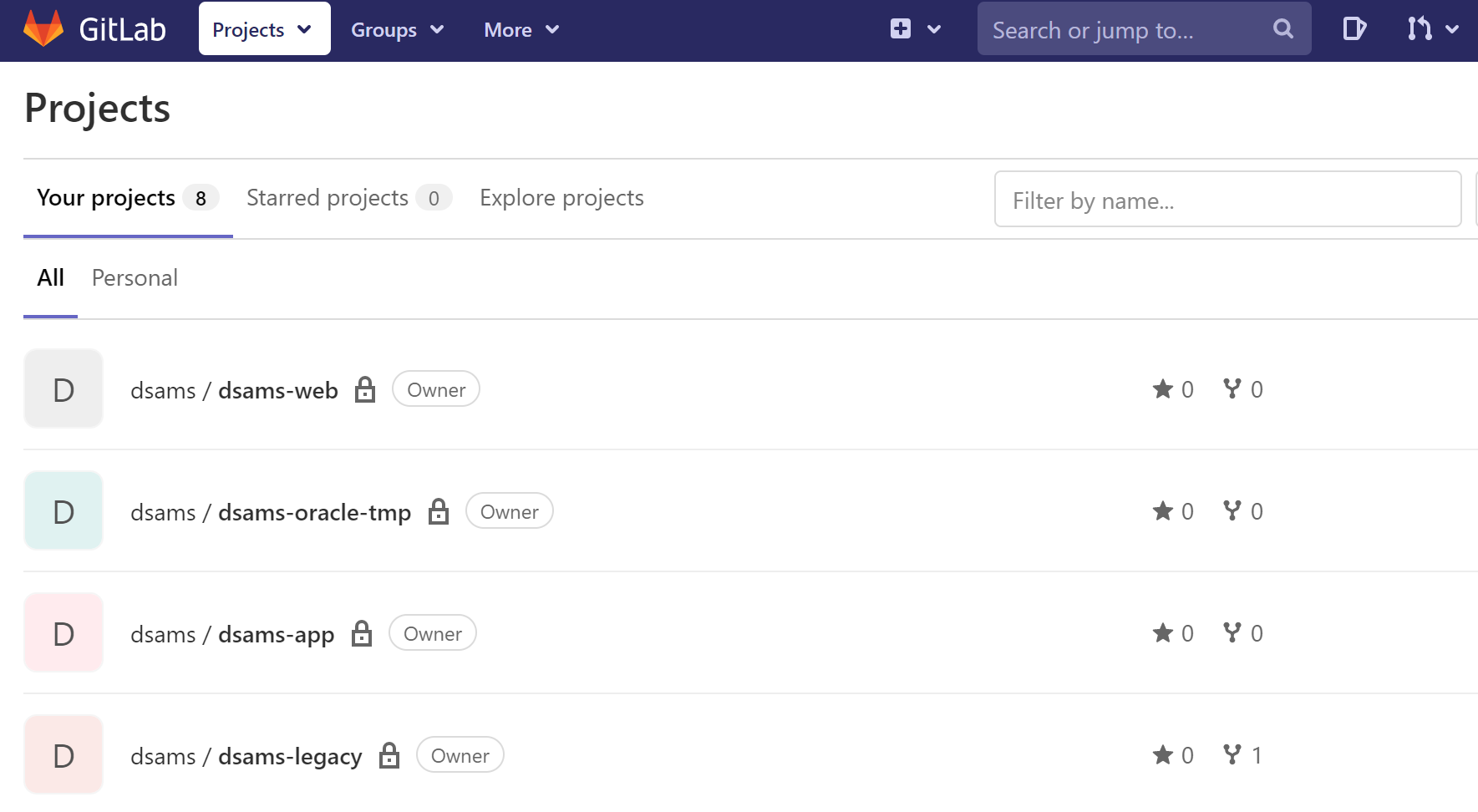


* 1. Branch and Feature

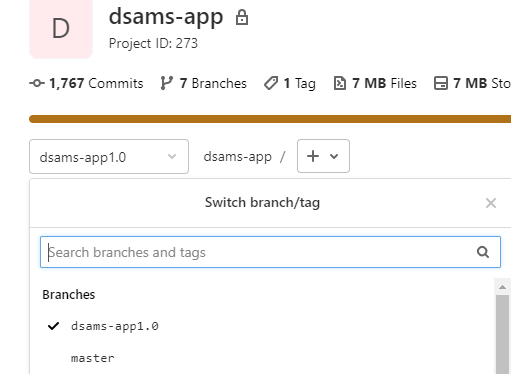
Currently DSAMS Web has three branch base on the release version

1. Development Branch: DSAMS-Web1.0
2. Integration Testing Branch: DSAMS-Web2.0
3. Production Branch: Master

DSAMS Source Code in GitLab Diagram



DSAMS Branches



1. **Source Code Review**
   1. Code Review

The code is reviewed by the developer team members when we have the technical team meeting.

* 1. Junit Test & Katalon

1. Each function/procedure in DSAMS-App will create an associate Junit test to make sure the function/procedure is working properly after deploy to the Tomcat Server.
2. The Katalon tool automate testing the DSAMS-Web from the Browser.
   1. SonarQube

The SonarQube find the vulnerabilities and inspect of code

Quality

* 1. Twistlock

Protect DSAMS applications across the entire container life cycle:

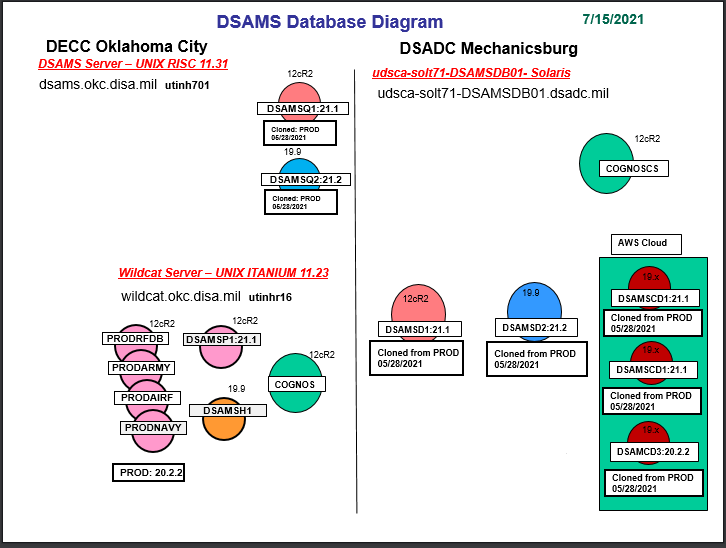
Build, Deploy, Runtime. Secure your clusters, nodes, container

engine, and other infrastructure components.

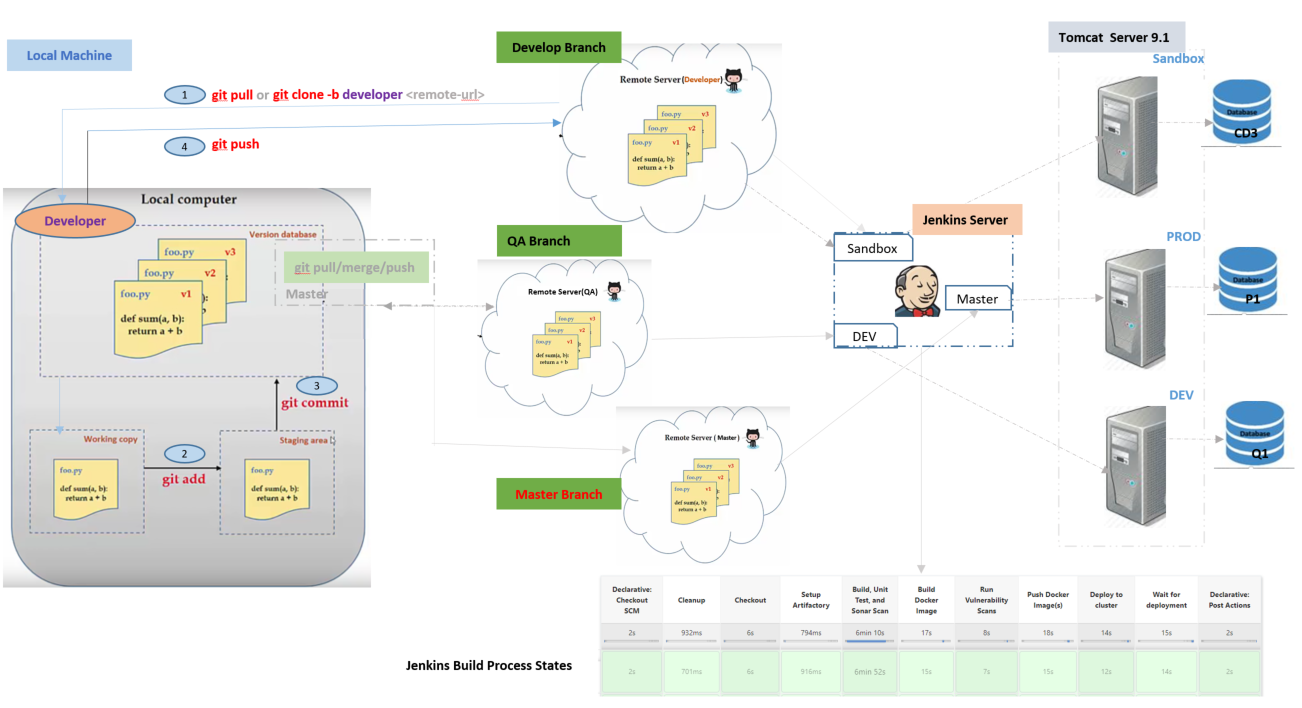
1. **Database View**

Currently DSAMS Web is using the DSAMSCD3 in AWS which is clone from the DSAMSD1 in on premise.

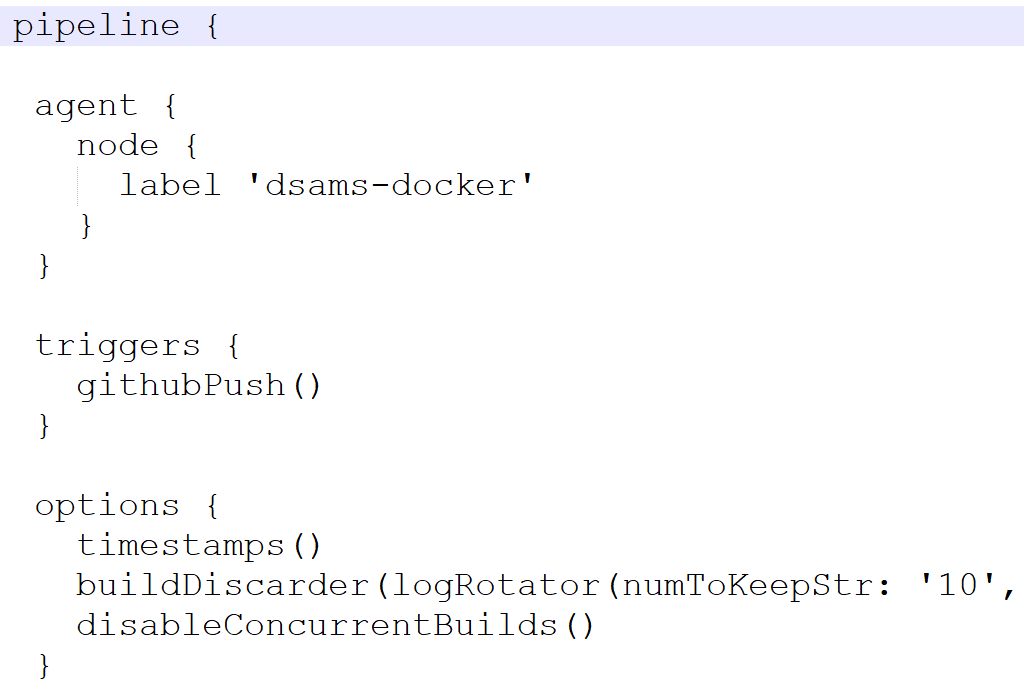
1. DSAMS Database Diagram on Premise.



1. For more details information, Please refer to DSAMS DBAs.
2. **Deployment View**
   1. Jenkins Build Process States Diagram



* 1. Jenkins (Jenkinsfile)



* 1. Docker (Dockerfile)



1. **Reference**

a) US Web Design System: <https://designsystem.digital.gov/>

b) 18F Accessibility: <https://accessibility.18f.gov/>